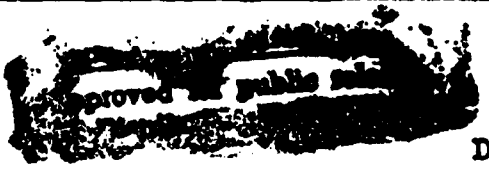
AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

THESIS

Richard Charles McEachin
Captain, USAF

AFIT/GST/ENS/94M-08



DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GST/ENS/94M-08

AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

THESIS
Richard Charles McEachin
Captain, USAF

AFIT/GST/ENS/94M-08

Approved for public release; distribution unlimited

AFIT/GST/ENS/94M-08

AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Richard Charles McEachin, B.S.

Captain, USAF

March, 1994

Approved for public release; distribution unlimited

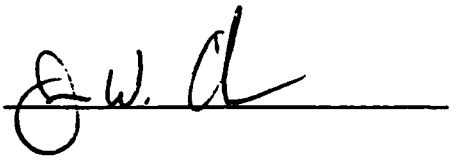
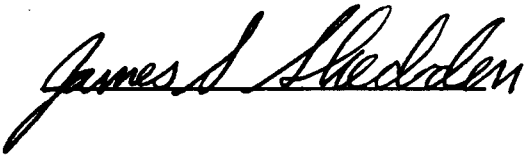
THESIS APPROVAL

STUDENT: Richard C. McEachin, Capt, USAF CLASS: GST-94M

THESIS TITLE: AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

DEFENSE DATE: 24 February 94

COMMITTEE:

Name/Title/Department	Signature
Advisor: James W. Chrissis, PhD, P.E. Associate Professor of Operations Research Department of Operational Sciences School of Engineering	
Reader: James S. Shedden, Lt Col, USAF, PhD Assistant Professor of Operations Research Department of Operational Sciences School of Engineering	

Acknowledgements

I would like to express my appreciation to Dr. Vipperla B. Venkayya. As my sponsor, he provided me with background on the structural optimization process, but more important, he helped me see the value of doing research and writing a thesis. Dr. James W. Chrissis advised me to look at all those pesky details that would have been so easy to ignore, and so, guided me to produce my best quality product. Lt Col James S. Shedden read my work, and asked me questions which were difficult to answer. In the process, he led me to the heart of the Simulated Annealing process. Professor Lester Ingber developed the Adaptive Simulated Annealing code that was the key to my applying Simulated Annealing to structural optimization. If I was building the code myself, I would have had a soap-box derby entry, instead he let me use his Corvette.

I appreciate the help of my parents, Angus and Madeline McEachin, who gave me a happy home to grow up in, three squares (at least), great siblings, and a quality education. Most important of all, I appreciate Neil, Danielle, and Aaron. My three children did mountains of dishes, vacuumed acres of carpet, and listened to me whine about all the hard work. I'm proud of what nice people they are, and I appreciate their patience and understanding. They made it possible for me to do this work.

Richard Charles McEachin

Table of Contents

List of Figures

List of Tables

Abstract

This thesis investigates the feasibility of using Simulated Annealing (SA) in structural optimization problems. SA is an optimization process modeled after the physical process of annealing. It is under development in many branches of the sciences, mathematics, and engineering. Its usefulness depends on making many iterations in a random search among the feasible values of the variables of a cost function. These many iterations are becoming more practical as computers become more powerful. This algorithm differs from other random search methods by accepting a worse solution (less optimal) than the previous, with a probability derived from a Boltzmann distribution. The probability of accepting a worse solution starts out high, and decreases as the iterations progress. This allows the algorithm to escape from a local optimum and, with a number of iterations approaching infinity, to find the global optimum.

The investigation involves solving benchmark structural optimization problems with an SA algorithm, and comparing its solutions to the solutions found by four other optimizers. The primary comparison is in quality of solution. However, each optimizer has its strengths, and an effort has been made to highlight the strengths of SA in structural optimization.

Overall, the analysis shows that SA has limited applicability in structural optimization. Two primary factors were found to adversely impact the performance of the SA algorithm in these problems. These factors are high dimensionality, and high levels of constraint. The difficulty involved in solving these problems with a random search increases exponentially with the number of dimensions. This limits the applicability of this algorithm to problems having less than twenty variables. The number, and non-linearity, of the constraints also have an appreciable effect on the success of the algorithm. A Measure of Complexity was created to quantify the combined effect of dimensionality and level of constraint. This measure can be used to predict the applicability of the SA algorithm in optimizing a given system of non-linear equations.

AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

I. Introduction

1.1 Background of the Problem

1.1.1 Engineering Optimization. The concept of optimization in engineering applications is centuries old. One of the first recorded engineering optimizations was by Galileo Galilei, when he found the strongest cantilever beam in bending and constant shear (Kamat, 1993:2). Typically, in formulating the optimization problem, the performance of a system is modeled by an *objective function.* The optimization task is to minimize or maximize the value of the objective function by finding optimal values of the *decision variables.*

1.1.1.1 Gradient Search. Many of today's more prevalent optimization methods depend on the calculus, developed in the 17th century. In these methods, the derivatives of the objective function (with respect to each of the decision variables) are set to zero, generating a set of simultaneous equations. These simultaneous equations are solved, yielding the values of the decision variables at the stationary point, or points (Kamat, 1993:3). The derivatives of the objective function form the *gradient* at a certain point, so this process may be termed a *gradient based search.* In a variation of this method, the gradient of the "surface" at a chosen point in the feasible region is found. With this information, an iterative gradient search algorithm can proceed from the chosen point, in a direction which always yields an improved solution at the next step. Some potential difficulties exist with a gradient search. Computing the gradients of the surface requires many calculations, and may be very time consuming. The search algorithm could stall at a relative minima (*local* optimum), rather than finding the overall minimum (*global* optimum) for the entire system. Also, the surface being searched must be continuous. In spite of these potential difficulties, gradient search has been successfully applied to many

optimisation problems. This basic process identifies the stationary points (function maxima, minima, or saddle points) in an *unconstrained* optimisation. In this case, there are no restrictions on the values of the decision variables.

1.1.1.2 Constrained Optimization. For most engineering applications where the decision variables represent physical characteristics (such as temperature, pressure, length, etc.), there are physical or economic constraints on the values of the decision variables. The constraints may be inequalities, equalities, or (upper/lower) bounds (also called side constraints). These constraints can be met in the optimization process by forming the augmented *Lagrangian* function. In forming the Lagrangian, each of the constraints is assigned a weighting variable (Lagrange multiplier) and the augmented function is a composite of the objective function and the sum of the weighted constraints. This augmented function becomes the objective function, and is minimized as before, including solving for the values of the multipliers. The use of multipliers increases the number of variables (dimensionality) of the problem, and so, the complexity of the minimization. However, the Lagrange multipliers make the connection between the objective function and constraints, and they give a proportional weight to the relative importance of each constraint. These methods are implemented currently in structural optimization by the steepest descent, conjugate gradient, sequential quadratic/linear programming, or feasible directions methods (Kamat, 1993:4).

1.1.2 Numerical Optimization Methods. With the advent of modern digital computers numerical methods of optimization became practical, with finite element, finite difference, and other discrete methods of optimization coming into use for structural optimization (Venkayya, 1993:4). Typically, the process involves five steps:

1) find a feasible solution

2) evaluate the solution

3) compare the new solution to previous solutions

4) accept (or reject) the new solution

5) stop when some final stopping criteria is met, or go to step 1.

1.1.2.1 Limitations of Numerical Optimization. While this type of search seems like a simple process, it has some potential pitfalls. For instance, non-linear systems may have more than one optimal solution. This could lead to finding a local optimum, rather than the single best global optimum over the entire search space. In the case of finding a local optimum, the particular local optimum found might depend on which initial feasible solution was chosen.

At each iteration, the method used for finding the new feasible solution may be the factor which has the greatest effect on the conduct of the minimization, and perhaps the quality of the final solution. For example, making a random perturbation of the system at each iteration is a very simple way of choosing the next solution, but this method may never find the optimum. On the other hand, a pure gradient search may lead to choosing the first local optimum encountered, rather than the global optimum (McLaughlin, 1989:26).

The most appropriate *stopping criteria* for a numerical search algorithm depend on the nature of the problem. For a convex objective function with concave constraints, a unique global minimum is found when the necessary and sufficient conditions are met. For non-convex problems, meeting the necessary conditions may not uniquely solve the problem (Kamat, 1993:5). Linear approximations of the non-linear functions can simplify the process, but then the overall quality of the solution depends on the quality and locality of the approximation.

Numerical search techniques are especially useful because of their generality. However, for problems with many variables, or for highly non-linear systems where convergence to an optimal solution is questionable, the numerical search process may stall. In these cases, potential solutions can be compared to optimality criteria, eliminating solutions that do not meet the necessary conditions.

1.1.3 Structural Optimization is the process of finding the best structure for a given task (Vanderplaatz, 1984:1). The structure has to meet all the requirements, as well as meeting the definition for being the "best." Being best may involve being the lightest, or the cheapest, or the smallest. In the case of modern aircraft, the best structure is often the lightest.

Structural optimization related to aircraft design is normally done to minimize weight. This is important for fuel conservation, and allows for maximum performance of the aircraft as a whole. In most current applications this minimization is accomplished by a gradient search of the objective function with constraints formed into a Lagrangian function. One example of an aircraft structural optimization system is the Automated Structural Optimization System (ASTROS).

ASTROS is a structural design optimization program being developed by the Flight Dynamics Directorate of Wright Laboratories. ASTROS includes a module to minimize the weight of aircraft components while meeting all strength and flexibility requirements. The primary algorithm in the minimization module forms the Lagrangian of the objective function and constraints and uses a gradient search to solve this as an unconstrained minimization (Canfield and Venkayya, 1990:1038). First, the system is solved for the values of the Lagrange multipliers. As mentioned, the minimization process involves finding the derivatives of the active constraints. However, if the value of a Lagrange multiplier is zero, then the corresponding constraint has no "weight" (is not active), and the algorithm can save time by not calculating the derivative of the constraint. However, finding the derivatives of the active constraints can be time consuming (Canfield, 1993). For large, fully constrained problems, this may overwhelm even a supercomputer. Several approaches have been tried to overcome this problem, but none is completely satisfactory for these large, fully constrained problems (Canfield and Venkayya, 1990:1037).

1.1.4 Simulated Annealing (SA) is an optimization process that does not use derivatives in minimizing a function. For this reason, SA may be useful in structural optimization to solve some problems which cannot be solved using a gradient search approach.

SA is a fairly new process for numerical optimization of many classes of problems. It is modeled after the centuries-old annealing process for metal and glass castings. Manufacturers anneal castings to make them tougher, by reducing their internal energy (McLaughlin, 1989:28). There are strong parallels between Simulated Annealing and the physical process of annealing. In each case, a system of many variables is minimized.

SA uses many steps in a random search to find the optimum of the system. Other random search algorithms are prone to selecting the first local optimum encountered. However, SA has a feature that helps it find the global optimum rather than a local optimum. The many steps required in SA are possible with modern computers, and the more capable computers become, the more useful SA will be.

1.2 Conduct of the Research

1.2.1 Purpose of the Research. This research investigates the advantages and disadvantages of applying SA to structural optimization problems. It highlights the areas where SA shows potential for further use in structural optimization. The results of this investigation can be used to determine if further work should be done to implement SA in specific structural optimization packages by commercial and government users.

1.2.2 Problem Statement. The problem posed for this research is to implement SA in benchmark structural optimization problems, and analyze the performance of SA in this application. The analysis will identify strengths and weaknesses of SA in this application and provide guidance to potential users concerning the applicability of SA to structural optimization problems.

1.2.3 Organization of the Report. This thesis consists of five chapters, beginning with this introductory chapter. Chapter II highlights the pertinent structural optimization and SA literature that has been published to date. Chapter III outlines the theoretical background and development of SA and Adaptive Simulated Annealing, as well as explaining the benchmarking process used for this analysis. In Chapter IV, the results of optimizing the benchmark problems with SA and analysis of these results are shown. Finally, Chapter V gives a summary, recommendations for further research, and conclusions of the investigation.

II. Literature Review

2.1 Simulated Annealing (SA) in Structural Optimization

While no direct applications of simulated annealing to structural optimization have appeared in the literature yet, SA has been applied in a wide variety of other contexts. The SA algorithm has been developed over the last twenty years by integrating proven techniques from statistical mechanics with new insights into combinatorial optimization. Kirkpatrick, et.al., brought SA to the attention of scientists in many fields when they published *Optimization by Simulated Annealing* (Kirkpatrick, Gelatt, and Vecchi, 1983). Prior to this, SA was an obscure mathematical tool, and its applicability to other fields was not widely recognized (Collins, Eglese, and Golden, 1988). Kirkpatrick's article becar launching point for many investigations into the use of SA in fields as diverse as com architecture, geology, chemistry, and biology.

Kirkpatrick, et.al, identified the connection between statistical mechanics and combinatorial optimization. They explained that in annealing, statistical mechanics models the minimization of the internal energy (heat) of many molecules. Combinatorial optimization minimizes (or maximizes) the value of an objective function of many variables. They gave examples showing how statistical mechanics in annealing is analogous to combinatorial optimization in designing computer chips and in the classic traveling salesman problem.

2.2 Outline of the Standard SA Algorithm

The SA algorithm involves six fundamental steps (Aarts and Korst, 1990:12)

1. Choose a feasible starting point and temperature; then evaluate the objective function.
2. Randomly perturb the system by changing the value of one or more decision variables, while ensuring that their values remain feasible.
3. Evaluate the objective function at the new point.
4. Apply the acceptance criteria for the new solution.

5. Repeat steps 1 through 4 until the system is stable, then reduce the temperature according to the annealing schedule.

6. Repeat steps 1 through 5 until the final stopping criteria is met.

2.3 Highlights of the Algorithm

(1) The first step in the algorithm is to choose a starting configuration and control parameter (analogous to *temperature* in physical annealing), then find the initial value of the objective function. While these choices of starting solution and temperature are unique to each application, SA is normally fairly insensitive to the starting conditions. In the application to structural optimization, this step establishes the initial physical characteristics of the structural components, ensures that all constraints are met, and determines the initial weight of the structure.

The term *temperature* is a holdover from the physical process of annealing, where it refers to the actual heat content of a casting. In simulated annealing, temperature is a parameter that controls the probability of accepting a new solution that is "worse" than the old one. The higher the temperature, the greater the chance of accepting a "worse" solution. This probability of accepting a worse solution is the feature that allows SA to leave a local minimum and continue to search for the global minimum. McLaughlin suggests that the initial temperature should be high enough that virtually any feasible solution will be accepted (McLaughlin, 1989:32), but a more common initial target acceptance rate is about 80 percent.

(2) The second step in the algorithm is to randomly perturb the system. In explaining combinatorial optimization, Kirkpatrick, et.al., described a random search method that accepts only lower values of the objective function at each iteration. It usually gets stuck in the local minimum closest to the starting point. This algorithm is often called the Greedy Algorithm because, in its "greed" to find any optimum, it will likely miss the global optimum and accept a local instead (McLaughlin, 1989:25). In 1985, Cerny presented a Monte Carlo algorithm to find approximate solutions to the traveling salesman problem. "The algorithm generates randomly the permutations of the stations of the traveling salesman trip,

with probability depending on the length of the corresponding route" (Cerny, 1985:41). This offers one method for generating random perturbations to a system. In structural optimization, this step corresponds to a random change in the physical dimension of one or more components.

(3) The third step is to evaluate the new solution. The specific mechanics of this evaluation depend on the application. For structural optimization, this step determines the total weight of the structure with the new dimensions.

(4) In the fourth step, accept or reject the new solution. If the new solution gives a lower value for the objective function, accept it. However, if the new solution gives a higher value, consider accepting it. This possibility of accepting the "worse" solution gives the SA algorithm the ability to leave a local optimum, and continue to search for the global optimum. This is the key feature that sets SA apart from other random search algorithms. From statistical mechanics, Kirkpatrick, et.al., described the Metropolis procedure to overcome the Greedy Algorithm's problem of stalling at a local optimum. "The Metropolis procedure from statistical mechanics provides a generalization of iterative improvement in which controlled uphill steps can also be incorporated in the search for a better solution" (Kirkpatrick, Gelatt, and Vecchi, 1983:674). This makes it possible for the algorithm to climb out of a local minimum and find a better local minimum, or the global minimum. Control for the uphill steps is given by the Boltzmann distribution (Figure 2.1):

$$Pr(E) = \frac{1}{Z(T)} \exp\left(\frac{-E}{k_B T}\right)$$

where $Pr(E)$ is the probability of accepting the uphill step, $Z(T)$ is a normalizing factor depending on the assigned temperature (T), E is the average energy level, and k_B is the Boltzmann constant. The value of k_B is a natural constant, determined by experimentation, which adjusts the shape of the Boltzmann distribution to model the physical annealing process. It normally would not represent a valid constant in the SA process, but a different constant may be appropriate. For a given change in temperature, when the temperature is high, the probability of accepting an uphill step is high. As the temperature is reduced, the probability of accepting the uphill step is reduced.

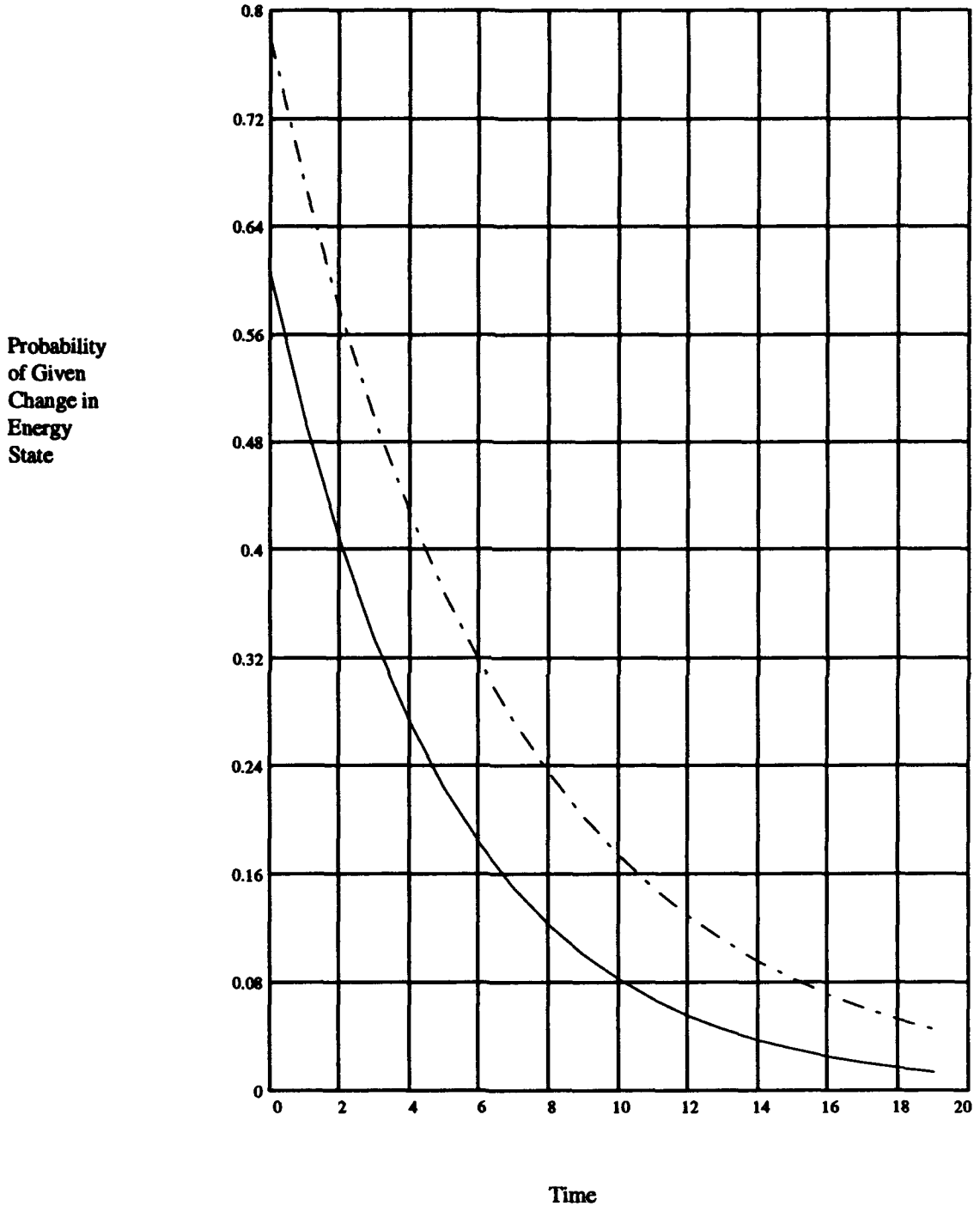


Figure 2.1 The Boltzmann Distribution at High and Low Temperatures

(5) The fifth step in the algorithm is to iterate at a given temperature and, when the system is at a stable average configuration for that temperature, reduce the temperature according to the annealing schedule. This schedule for reducing the temperature is critical to the success of either real or simulated annealing. According to Cerny, "Experiments ... are done by careful annealing, first melting the substance, then lowering the temperature slowly, and spending a long time at temperatures in the vicinity of the freezing point. If this is not done, and the substance is allowed to get out of equilibrium, the resulting crystal will have many defects" (Cerny, 1985).

Quenching is the process of deliberately reducing the temperature quickly, without allowing the substance to reach equilibrium. This degenerates the SA algorithm to an ordinary random search like the Greedy Algorithm. In annealing, this process creates a brittle casting, but it is much quicker, and in some cases may be preferred to the slow annealing process. Quenching is not normally used in SA. To get the lowest possible cost with SA, the annealing schedule must allow the system to reach steady-state at each temperature. On the other hand, spending too much time at a given temperature wastes computer resources. So, the annealing schedule must allow the system to stabilize before changing temperature, and then change promptly.

The cooling schedule is often found by trial and error (Brooks and Verdini, 1988:431). However, Basu and Frazer suggest that it may be cost effective to spend up to 80 percent of the total CPU time to establish the best cooling schedule (Basu and Frazer, 1990). Collins, et.al., listed five different schemes for controlling the temperature, T, (Collins, Eglese, and Golden, 1988):

- A constant value of T; $T(t) = C$
- An arithmetic function of T; $T(t) = T(t-1) - C$
- A geometric function; $T(t) = a(t)T(t-1)$
- An inverse; $T(t) = C/(1+t\alpha)$
- A logarithmic function; $T(t) = C/ln(1+t)$

(6) The last step in the SA algorithm is to iterate until the stopping criteria is met. Several classes of stopping criteria can be used (Collins, Eglese and Golden, 1988).

- In the simplest criteria, a fixed amount of CPU time is allocated, and the process stops when the time runs out (Brooks and Verdini, 1988:429).
- Another approach is to compare the value of the objective function at each iteration with the value at previous iterations. Under this criteria, stop when the function reaches a stable value for a certain number of iterations (Brooks and Verdini, 1988:430).
- If there is a certain target value of the function (a known or estimated minimum), stop when the configuration meets the target (Brooks and Verdini, 1988:430).
- When the algorithm is near the optimum the ratio of accepted configurations to total configurations will become very small. The algorithm can stop when this ratio reaches a predetermined value (Ingber, 1993b:6).
- If none of the other criteria are met, stop when the temperature reaches a value near zero (Collins, Eglese, and Golden, 1988:212). At this point the algorithm degenerates to a random search, and the cost of further annealing should be compared to the benefit that might be gained.

When the correct stopping criteria is met, the algorithm will have a solution close to the global optimum.

2.4 New Developments

Until about 1987, simulated annealing was considered useful for finding only local optima. However, new references in the field refer to SA finding global optima rather than local optima. Brooks and Verdini report: "The rapid increase in inexpensive computing power has contributed to an increase in interest in solving *global* optimization problems using stochastic methods (Brooks and Verdini,1988:427)." Given enough iterations, SA can be expected to find the global optimum. With a powerful enough computer, this may be realistic for all discrete optimizations and for many continuous functions.

Some new methods are being investigated to help speed up the SA process. Goldstein and Waterman have considered varying the size of the neighborhood of the acceptable steps by changing the permutation mechanism (Goldstein and Waterman,1988:411). Tovey wrote *Simulated Simulated Annealing* to discuss three other methods of speeding the process (Tovey, 1988). He calls them the Surrogate Function, Neighborhood Prejudice, and Target Prejudice *swindling* ideas. These have potential for improving the speed of SA by making each iteration more efficient. This suggests the need to count the number of iterations that SA uses for each solution. An old method for speeding SA is to make it interactive with a human operator. This method is gaining new attention as interaction techniques are improved. With a human to intervene when the process takes a bad turn, SA's efficiency can be greatly improved (Tovey, 1988:405). New parallel processing hardware will also make SA more attractive by increasing the speed of processing when making many iterations and when checking the constraints imposed on the solution.

Overall, the literature on Simulated Annealing has touched many different fields. The algorithm is under rapid development by many researchers. As yet, the application of SA to structural optimization problems has not appeared in the literature. However, the foundation of SA in general has been thoroughly developed, opening the door to research on this particular application.

III. Simulated Annealing and Benchmarking

This chapter outlines the basic tools used to evaluate the application of Simulated Annealing to structural optimization problems. It highlights the theoretical development of the standard Simulated Annealing algorithm and the features specific to Adaptive Simulated Annealing, the software package implemented for this research (Ingber, 1993b). The concept of benchmarking is introduced, and the specifics of the benchmark problems are discussed.

3.1 Theoretical Development of the SA algorithm

The standard SA algorithm uses a Boltzmann distribution to control the probability of accepting a worse solution at each iteration. As discussed in Chapter 2, the Boltzmann distribution is a negative exponential with parameters to control the shape of the curve, thus controlling the probability of accepting a worse solution. Each time a worse solution is proposed, the value of the Boltzmann distribution is compared to the value of a random variable drawn from a uniform distribution. If the Boltzmann value is higher than the uniform random variable, the new solution is accepted, even though the cost (weight) is higher. At each iteration, there is some finite probability of accepting a worse solution. However, this probability approaches zero near the end of the program, as the temperature approaches the "freezing" point.

The primary control parameter in the Boltzmann distribution is the temperature. This term initially referred to the actual temperature in a physical annealing process, and so was held over by most users of SA. In Simulated Annealing, "temperature" refers to a control parameter for the negative exponential distribution, even if it looks different from the original Boltzmann distribution. Because the temperature term is in the denominator of the negative exponential term, a high temperature yields a large value from the Boltzmann distribution, and a high probability of accepting a worse solution. This is the case at the beginning of the process. As the temperature is decreased, the value drawn from the distribution decreases, and the probability of accepting a worse solution decreases.

3.1.1 Global Optimality. It has recently been proven that, with an infinite number of iterations, Adaptive Simulated Annealing will find the global optimum (Ingber, 1993a:39). Since there is always a finite probability of accepting a worse solution at each iteration, the algorithm can always leave a local optimum. As the number of iterations approaches infinity, Adaptive Simulated Annealing must leave all of the local optima and find the global optimum. At each iteration of the algorithm, Adaptive Simulated Annealing saves the *best cost yet*, and the global optimum will be saved even if the algorithm leaves this configuration.

3.2 The Adaptive Simulated Annealing Algorithm

Adaptive Simulated Annealing has been developed and modified over the last ten years by Dr. Lester Ingber (Ingber, 1993b). The code was originally developed and made available in 1987 as Very Fast Simulated Reannealing, and is under continuing development.

3.2.1 Temperatures. In Adaptive Simulated Annealing, a temperature term is used to control the step size from one solution to the next proposed solution as well as to control the probability of accepting a new "worse" solution. The temperature which controls the acceptance probability is called the *cost temperature* and the control for the step size is the *parameter temperature* (Figure 3.1). If a feasible solution is available, a new proposed solution is generated from it. The new proposed solution is projected uniformly along each variable (dimension) within the upper and lower bounds defined for that variable. When the parameter temperature is very high, there is a high probability of choosing a value far from the current value for each variable. At this high temperature almost any value within the bounds defined for the variable is likely to be chosen for the new proposed solution. When the parameter temperature is reduced, there is a lower probability of choosing a value far from the starting value for each variable. This allows the algorithm to survey the entire solution space, then close in on the optimum. If there is no initial feasible solution, the parameter temperature remains high and proposed solutions are created by randomly choosing variable values within the upper and lower bounds defined for each variable, until a feasible solution is found.

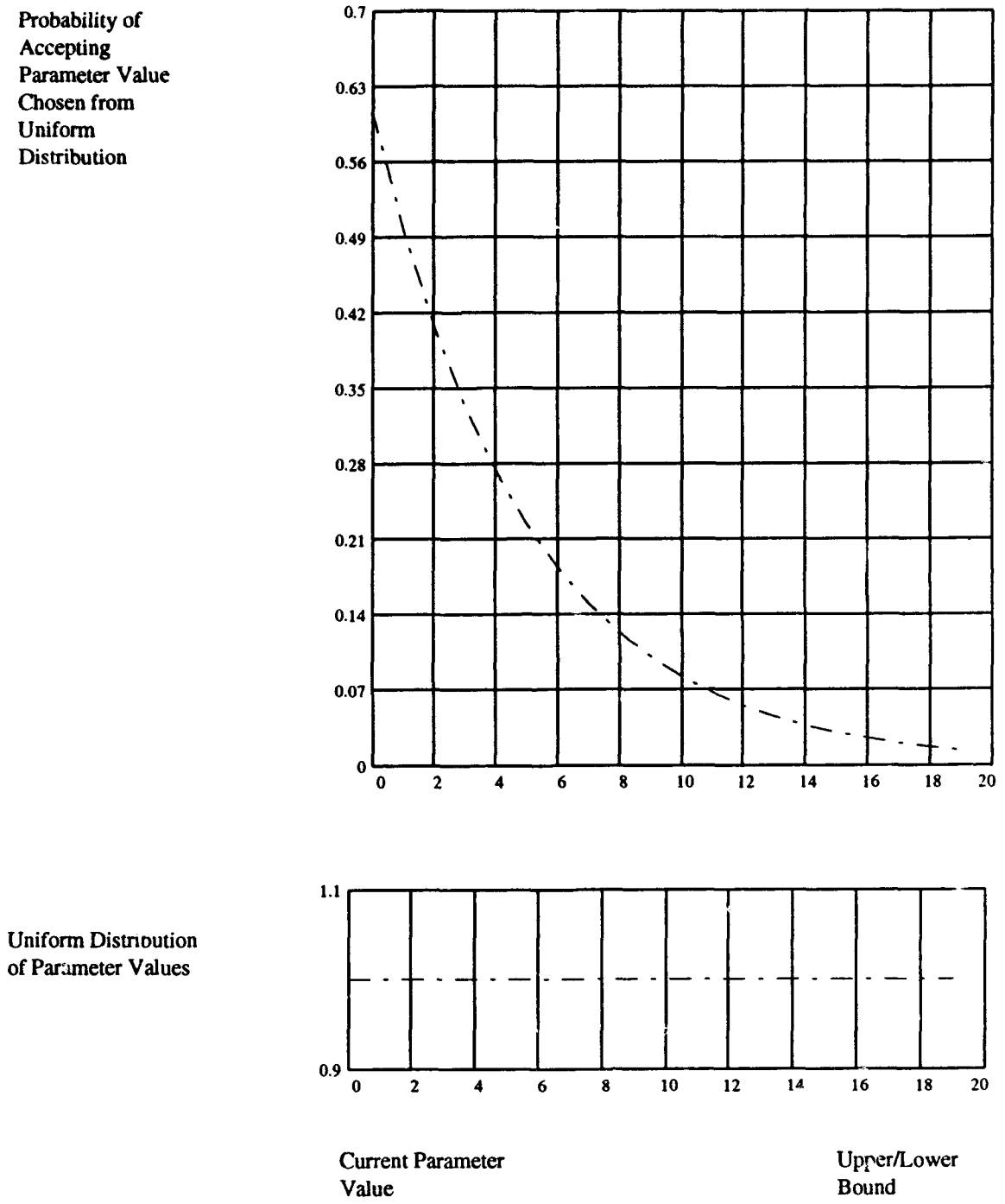


Figure 3.1 Parameter Temperature Compared to Uniform Distribution

3.2.2 Fundamental Distributions and Equations (Ingber, 1993b:5). There are five fundamental equations which control the function of Adaptive Simulated Annealing.

- The *Parameter Generating PDF* generates new variable values:

$$p(k+1) = p(k) + y(B - A)$$

where $p(k)$ and $p(k+1)$ are the current parameter value and the next for the variable representing one dimension. The y parameter is a uniform $[-1, 1]$ random variable. The B and A terms represent the upper and lower bounds defined for the variable.

- The probability of using this newly generated parameter value in the new proposed solution is controlled by the *parameter temperature* function:

$$g(y, T_p(k_p)) = \frac{1}{2[|y| + T_p(k_p)] \ln[1 + \frac{1}{T_p(k_p)}]}$$

where $T_p(k_p)$ is the parameter temperature at iteration k_p defined in the algorithm according to the annealing schedule created by the *Annealing Temperature Schedule*. This is the behavior described previously: a high parameter temperature gives a high probability of making a large step, and a low temperature gives a low probability of making a large step.

- The *Parameter Annealing Temperature Schedule* controlling the parameter temperatures is given by:

$$T_p(k_p) = T_p(0) \exp(-W k_p^{1/D})$$

where $T_p(k_p)$ and $T_p(0)$ are the current temperature and the initial temperature, respectively, W is the current weight, k_p counts the number of iterations completed in selecting values for this parameter, and D is the number of dimensions. Note that as the weight, W, is reduced, the temperature tends to rise. However, the number of iterations increases steadily, and reduces the temperature. It is this balance between

the weight and the number of iterations that drives the annealing schedule for the temperatures of the parameters.

- The *cost temperature* (Acceptance PDF) controls the probability of accepting worse solutions:

$$F = \exp\left(\frac{-[W(k_c) - W(k_c - 1)]}{T_c(k_c)}\right)$$

where $W(k_c)$ and $W(k_c - 1)$ are the weights (or costs) of the structure at the current and previous solutions, and $T_c(k_c)$ is the assigned cost temperature according to the *Annealing Temperature Schedule*. This value is compared to a uniform [0,1] random variable U (Figure 3.1). If $F > U$, the new solution is accepted; if $F \leq U$, another solution is tried.

- The cost temperature is reduced according to the *Cost Annealing Temperature Schedule*:

$$T_c(k_c) = T_c(0)\exp(-Wk_c^{1/D})$$

This cost temperature schedule is similar to the parameter temperature schedule except that k_c counts the number of iterations tried for the cost function.

3.2.3 Initial Conditions and Stopping Criteria are required for the standard SA algorithm as well as for Adaptive Simulated Annealing. With the exception of the reannealing parameters and the specific names of the other parameters, the terms used here can be applied to the standard SA algorithm. The initial conditions of the optimization determine the initial feasible solution, initial temperatures, and user selectable features. Some of these user-controllable features are:

- *Initial Cost Temp* can be user defined to adjust the initial rate of acceptance.
- *Include Integer Parameters* can be set to make Adaptive Simulated Annealing find integer solutions only (Integer Programming).

- *User Initial Parameters* can be set to give Adaptive Simulated Annealing a user defined initial feasible solution.
- *User Initial Parameters Temps* can be set to adjust how fast Adaptive Simulated Annealing initially surveys the solution space.

The stopping criteria define when Adaptive Simulated Annealing stops making iterations, and considers reannealing, or (for SA in general) stops annealing at the final solution. The process stops when any one of the selected stopping criteria is met, so the stopping criteria plays an important role in determining the precision of the solution. Again, these criteria apply to generic SA as well as to Adaptive Simulated Annealing. Some of these tolerances are:

- *Limit Acceptances* defines the maximum number of feasible solutions Adaptive Simulated Annealing accepts before stopping.
- *Limit Invalid Generated States* defines the maximum number of infeasible solutions Adaptive Simulated Annealing generates before stopping.
- *Accepted to Generated Ratio* defines the smallest allowable ratio of accepted to generated solutions. When the algorithm is close to the minimum, the cost temperature may be low, so few new solutions will be accepted and this ratio approaches zero.
- *Cost Precision* defines the smallest difference between cost (weight) values of the objective function considered to be significant. This influences the use of *Maximum Cost Repeat*, which stops the iteration process when the cost (weight) is repeated a defined number of times.
- *Parameter Temp Test* sets a flag when the parameter temperature is reduced to a defined value. When the parameter temp is near zero, only very small steps are likely to be made from one feasible solution to the next, and it is relatively expensive to continue searching compared to the potential benefit.
- *Cost Temp Test* sets a flag when the cost temperature is reduced to a defined value. When the cost temperature is very low, there is a very low probability of accepting a new higher cost, indicating that the search is near the optimum. At this point the

algorithm is an ordinary random search, accepting only improved solutions at each iteration.

3.3 Features Specific to Adaptive Simulated Annealing

This code is available free from Dr. Ingber. It is made to be highly portable for platforms ranging from microcomputers to supercomputers. It is written in the "C" programming language, which is modern and flexible, and the code is setup for ANSI C or for previous versions. Dr. Ingber himself is available for consultation by electronic mail. In fact, he provides feedback to users and updates to the code via a mailing list for anyone interested in the code. This communication among users creates the interaction among users which is necessary for research progress.

An important underlying reason for using the Adaptive Simulated Annealing code for this investigation is its many user-controllable options. These options make the code flexible and allow the user to target the specific features of SA which are needed for a specific task. Adaptive Simulated Annealing implements the basic SA algorithm, and makes it very user friendly. Default values for each of the options are built into the Adaptive Simulated Annealing algorithm. Some of these options that are specific to the Adaptive Simulated Annealing algorithm are:

- *Reannealing.* In the reannealing process, when the initial stopping criteria is met the parameter temperatures are rescaled and the annealing is restarted. This allows greater precision in the solution by minimizing round-off error. Reannealing is an optional process, selectable by the user, along with the number of restarts to be made.
- *Self Optimizing.* In case the initial conditions are difficult to define, the Adaptive Simulated Annealing algorithm will anneal the annealing process. This process optimizes the initial conditions for a subset of the solution space, then provides these initial conditions for the full solution. This process is CPU-intensive, but it gives the program a starting point in case the defaults don't work and the user has no other method of identifying initial conditions.

- *Curvature Zero* tells Adaptive Simulated Annealing not to calculate the slope of the solution space at the optimal point. This calculation is used when rescaling the parameters based on curvature, prior to reannealing.
- *Quench Parameters* allows the annealing process to proceed to a defined point, then the parameter temperature is reduced to a value near zero. This effectively stops the annealing process at that point, and the search degenerates to an ordinary random search. This option allows for a quick solution in the case of many dimensions or a complex system of equations to evaluate. However, the solution found is very approximate and the theoretical guarantee of global optimality is lost.
- Adaptive Simulated Annealing has an option to keep track of *CPU-time used* in finding a solution. This makes it easy to compare the performance of the algorithm with standard benchmark problems on a standard computer platform.
- The *input/output* functions are very flexible. This feature is especially user friendly. The user can easily define the input and output options, as well as the name of the output file and the media where it's written.
- To make the code applicable to many computer platforms, the machine precision, and overload values are user selectable.

3.4 Benchmarking in Structural Optimization

Structural optimization is carried out using various optimization packages. Typically, to compare the performance of one optimizer against others, standardized "benchmark" problems are run with each optimizer under conditions as similar as possible. The benchmark problems should give a representative sample of the types of problems that the optimizers can be expected to solve, as well as a common ground for comparison among the optimizers.

The relative quality of the solution found, the number of iterations and the time required to find the solution can be compared among the optimizers. The relative quality of the solution is the easiest of these to compare; normally the lowest value that meets the constraints is the best solution. Other criteria are harder to compare among optimizers.

The number of iterations required depends on the design of the algorithm, and may depend on the definition of an "iteration" for a certain algorithm. The time required for a certain optimizer to find the solution varies depending on the type of problem, the starting point, the computer architecture, and the precision required.

Many structural optimization problems can be formulated as standard linear or non-linear programs, with the objective function and constraints explicitly defined at the start of the program and evaluated at each iteration. However, because of the complexity of evaluating highly non-linear functions and their derivatives, even a supercomputer may be overwhelmed by problems with many variables. To resolve this shortcoming, large structural optimization problems are usually solved with some type of active-constraint strategy. In this formulation, the constraints are not explicitly defined at the beginning of the non-linear program. Rather, the active constraints are redefined at each iteration and only these constraints are evaluated. This process resolves some of the size and complexity difficulties in large problems, but requires the extra step of redefining the active constraints at each iteration.

The active-constraint formulation does not lend itself to SA, which evaluates all of the constraints at each iteration. For SA, this saves the effort of frequently redefining the constraints, but requires working with a larger non-linear program. However, since SA does not have to calculate the derivatives of the objective function and constraints at each iteration, a larger problem may be easier to evaluate and may not overtax the computer's capacity.

3.4.1 Benchmarking with SA. To evaluate the performance of SA in structural optimization, eighteen explicitly defined benchmark structural optimization problems were solved. These eighteen problems, along with the solutions found by four other optimizers, were provided by the Flight Dynamics Directorate at Wright Laboratories. These problems represent "real" structural optimization problems. They are formulated as linear or non-linear programs, and range in number of variables from two to eight, in number of constraints from one to twenty-five, and in complexity from linear to highly non-linear

(Venkayya, Tischler, and Pitrof, 1992). None of the eighteen structural optimization problems included more than eight variables.

To test simulated annealing on problems of higher dimensionality, two non-linear programs were chosen from another set of benchmark problems (Floudas and Pardalos, 1990). These are not "real" structural optimization problems, but they are formulated in a manner similar to the structural optimization problems. They have thirteen and twenty variables, respectively. Using these problems allowed a more complete test of the capacity of the Adaptive Simulated Annealing algorithm in solving problems with higher dimensionality.

3.4.2 Problem Formulation. Each problem is expressed in the following form:

Min

$$F(x)$$

Subject to

$$h(x) = 0$$

$$g(x) \leq 0$$

where $F(x)$ is a real-valued continuous function, $h(x)$ represents the set of equality constraints and $g(x)$ represents the set of inequality constraints. All decision variable values must be non-negative, since the variables in this set represent the physical dimensions of structures, and negative values would normally be meaningless.

3.4.2.1 Constraints. Except for upper and lower bounds, no method for evaluating constraints is explicitly built into the Adaptive Simulated Annealing algorithm. The set of bounds defines the neighborhood that the algorithm initially searches. If there are no other constraints, this represents the feasible region. However, structural optimization problems are usually constrained beyond any upper or lower bounds on the individual variables. In aircraft structural optimization, constraints must be met for flutter, displacement, total weight, etc., and the constraints may make the actual feasible region much smaller than the neighborhood set by the bounds on the variables.

To ensure that the constraints were all met prior to evaluating the objective function, each constraint was checked for feasibility. If all of the constraints were met, the objective function was evaluated, and this weight was returned to the optimizer. If any constraint was not met, a flag was set and no value was returned to the optimizer, saving the time which would have been required for evaluating the objective function for an infeasible solution.

Inequality Constraints. By far, most of the constraints in structural optimization problems are inequalities. This is because, in most structural optimization, a given performance level or physical characteristic must be met or exceeded in finding the best structure. In the formulation for optimizing these benchmark problems, all inequalities, $g(x)$, are expressed as $\leq$ constraints. This includes the cases where the original formulation included a function which was bounded above and below. In this case the bounds were rewritten as two separate inequality constraints. This constraint structure is simple. A formulation with fewer constraints might be possible at the expense of making each constraint more complex.

Equality Constrains. There is only one equality constraint in the entire set of benchmark problems. However, since equality constraints are possible in structural optimization problems, consideration was given to satisfying them. One way of satisfying these equality constraints is to use Lagrange multipliers and build the equality constraints and the original objective function into an augmented Lagrangian objective function. This method was not used because it increases the dimensionality of the non-linear program by introducing more variables (the Lagrange multipliers). Instead the equality constraint was used to redefine the value of one variable in terms of the others by *direct elimination* of one variable. This redefined term was then substituted back into the problem and the optimization was conducted without equality constraints. This process makes the inequality constraints more complex, but it reduces the dimensionality of the problem and the total number of constraints.

3.4.2.2 Objective Function. If the constraints are all met, the objective function is evaluated to find the total weight of the structure with the new values of the

variables. This value is returned to the optimizer, and is compared to the previous value of the objective function. If this new value is better than the previous value, the new solution is kept as the current solution. If it is not better than the previous, it may still be kept as the current solution with a probability depending on the current temperature.

3.4.2.3 Initial Feasible Solution. The algorithm accepts an initial feasible solution, if one is available. Since all of these problems have been solved by other optimizers, a very good initial feasible solution was available. As part of the evaluation, this solution was provided to the Adaptive Simulated Annealing algorithm in some runs. Additionally, in some cases, the known optimum was used to narrow the bounds of the neighborhood around each variable. These "unfair" variations in the initial conditions were used in cases where the Adaptive Simulated Annealing algorithm could not find a solution, and were considered in the analysis of the results.

3.4.3 Benchmarking with Other Optimizers. The other standard optimizers used on the benchmark problems represent basic techniques currently in use for structural optimization. FUNOPT uses a Generalized Compound Scaling scheme, ADS uses a Modified Feasible Directions approach, NEWSUMT implements a Sequential Unconstrained Minimization with Penalty Function, and NLPQL implements a Sequential Quadratic Programming scheme. All of the benchmark structural optimization problems were optimized by ADS, and NLPQL. Seventeen of them were optimized by FUNOPT, and only the first six were optimized by NEWSUMT (Venkayya, Tischler, and Pitrof, 1992:2). Solutions to the two additional non-linear programs were provided by Floudas and Pardalos (Floudas and Pardalos, 1990: 9-15).

Non-linear programs frequently have multiple local optima, and the solutions found to the benchmark problems by the various optimizers illustrate this point. Each non-linear program poses a unique problem for each optimizer, depending on problem structure, methodology, implementation, initial conditions, and other factors.

3.4.4 Measure of Complexity. The number of local minima, as well as the complexity of evaluating the objective function and constraints depend on the level of non-linearity

of these functions and the number of variables. A linear program has a linear objective function and constraints, and is one of the simplest to evaluate. A function with a non-linear objective function and constraints is much more difficult to evaluate, and may have many local optima. It became apparent during the course of this investigation that, along with the number of variables, the complexity of the constraints is an important factor in the success or failure of the simulated annealing algorithm in finding the global optimum. These factors determine whether or not an optimum is found, the time required to find an optimum, and the quality of the solution found.

The algorithm initially searches the entire neighborhood defined by the bounds on the variables. If there are many constraints, or they are complex, the actual feasible region may be much smaller than this neighborhood. In one case, the feasible region was estimated to be smaller than one millionth of the neighborhood. To illustrate how this affects the search, consider a two-dimensional neighborhood the size of this standard (8.5×11 inch) sheet of paper. A feasible region one millionth the size of this two dimensional neighborhood would be approximately $\frac{1}{100}$ of the size of this letter "O", even smaller than a period (".") on this page. In this case, the Adaptive Simulated Annealing algorithm starts a random search of the entire neighborhood (8.5×11 inches), and has a very small probability of finding the feasible region.

It seemed possible that some measure of complexity could be used to predict the effects that the constraints and the number of variables would have on an optimization by the Simulated Annealing algorithm. After a review of the literature in the field, and consultation with several mathematicians, it was determined that no such measure exists. An index representing a Measure of Complexity (MOC) was defined to help predict these effects. The objective function and the entire set of constraints for a given problem are evaluated and assigned a MOC value based on the non-linearity of the objective function and constraints, the number of variables, and the number of constraints. Based on the MOC value, a user may be able to make predictions about the performance of the algorithm on a certain problem. The MOC index was not designed to predict the performance of other optimizers. However, the same criteria used here would be appropriate to build indexes for other optimizers, or the MOC may apply as presented.

Measure of Complexity (MOC) Calculation. Sum the point values assigned to the objective function and constraints, according to the following criteria :

1 point for each term with a linear exponent or negative exponent

2 points for each term with an integer non-linear exponent or integer crossproduct

3 points for each term with a non-integer exponent

2 points for each constraint

3 points for each variable

If a term meets the criteria for more than one point value, assign the highest appropriate point value for the term.

Example:

Min

$$f(x) = 3x_0x_1x_2^{0.66} + 2x_2$$

Subject to :

$$g1 = x_0^2 - 1$$

$$g2 = x_0x_2 + x_1^{-1.0}$$

```
MOC count

          f(x):   3 points for non-integer exponent
                  1 point for integer exponent
     variables:   9 points for three variables
            g1:   2 points for non-linear exponent
            g2:   2 points for integer crossproduct
                  1 point for negative exponent
   constraints:   4 points for two constraints

                 -------
Sum MOC            22
```

Before each problem was optimized with the Adaptive Simulated Annealing algorithm, information was gathered on the number of variables, number and type of constraints, the Measure of Complexity, and variations in the initial conditions given to the algorithm. After optimizing, the best solution found, the number of solutions generated, and the amount of time required to find the solutions were noted. Chapter Four shows how this information was used to analyze the performance of the algorithm, and to make comparisons with the other optimizers.

IV. Results and Analysis

This chapter presents the results of optimizing each of the benchmark problems using the Adaptive Simulated Annealing algorithm. The analysis centers on three main results. The first concerns the effects of variations of the initial conditions and stopping criteria. Next, the quality of solutions found by the Adaptive Simulated Annealing algorithm is compared to the quality of solutions found by the other optimizers. And finally, the effects of high dimensionality and high non-linearity of the objective function and constraints are related to the success of the optimization.

4.1 Variations in Initial Conditions/Stopping Criteria

The initial conditions and stopping criteria given to an optimizer can have a great impact on the success of the optimization. These parameters may determine whether or not the algorithm finds a solution, the quality of the solution found, and the time required to find a solution. By trial and error, it was determined that the initial conditions/stopping criteria that seem to make the most significant difference for SA in structural optimization are: *Limit Acceptances, Limit Invalid Generated States, Accepted to Generated Ratio, Cost Precision, Maximum Cost Repeats, Number of Cost Samples, User Initial Parameters (true/false), Activate Reanneal (true/false), Upper/Lower Bounds, and Random Number Generator Seed.* Benchmark Problem One was used to evaluate the effects of variations in these initial conditions/stopping criteria on the success of the algorithm (Table 4.1).

To identify the effects of varying these ten parameters, Problem One was initially optimized using a set of "Standard Conditions" (page C.1). Each condition that was identified as having a potentially significant effect on the optimization was isolated and modified, then the optimization was re-accomplished for each change. Analysis of the effect of each variation on optimizing this problem was generally useful for predicting the effects of variations in other problems. No effort was made to evaluate interaction effects among these variations in the initial conditions/stopping criteria, because the results would be so specific to each individual problem.

Table 4.1 Effect of Variations in Initial Conditions and Stopping Crit ria

Modified Parameter	Standard Value	Modified Value	Solutions Generated	Solution Time
Standard Condition No Modif			33453	0m 21.53s
Limit Acceptances	1000	10000	769257	8m 21.53s
Lim Invalid Gen States	1000	10000	33453	0m 21.48s
Acc/Gen Ratio	10^{-4}	10^{-8}	33453	0m 21.47s
Cost Precision	10^{-8}	10^{-4}	33453	0m 21.33s
Max Cost Repeats	2	5	33453	0m 22.31s
No. Cost Samples	2	5	3155	0m 01.99s
User Init Parameters	Yes	No	29756	0m 20.09s
Activate Reanneal	No	Yes	14912	0m 09.14s
Bound Change	0 to 10	0 to 20	57787	0m 38.4s
Bound Change	0 to 10	0 to 100	0	No Feasible Solution
Random No. Seed	696969	697969	80091	0m 54.51s
Random No. Seed	696969	696971	3735	0m 02.55s

Benchmark Problem One was chosen to evaluate the effects of these variations because of its simplicity. It has only two linear constraints, one non-linear constraint, and two variables in the objective function (MOC value of 22). The Adaptive Simulated Annealing algorithm was able to find the same optimal solution under all but one of the variations, eliminating variability in the quality of the solution. This made the time required to solve the problem the primary basis for comparison of the variations. Generally, the time required is proportional to the number of solutions generated. As shown in Table 4.1, the effect of each variation can be seen in the results of these tests:

- *Limit Acceptances* limits the total number of feasible solutions accepted by the optimizer (page C.2). The effect of a change in *Limit Acceptances* is significant. By increasing the number of acceptances by a factor of 10, the number of solutions generated increased by a factor of 25, and the time required to find the solution also increased by a factor of 25. Analysis of intermediate steps in the optimization showed that the actual number of solutions generated and the time required to find the optimal values of the decision variables were similar to those required under the standard conditions. However, the algorithm continued to "spin its wheels" for the additional allowed number of solutions. To avoid this waste, the user can allow a relatively small number of acceptances initially, then increase the number allowed in subsequent optimizations if an improved solution is desired.
- *Limit Invalid Generated States* limits the number of infeasible solutions the optimizer may generate (page C.3). For this problem, changing this parameter did not influence the optimization significantly. This parameter could be significant in highly constrained problems, where many infeasible solutions will be generated at the beginning of the optimization. In that case, allowing more infeasible solutions could be the key to the optimizer finding a feasible solution. This was the case with Benchmark Problem Six (pages C.18 and C.19). The algorithm could not find a solution under the standard conditions, but when the stopping criteria allowed more infeasible solutions, the algorithm found a good solution.

- *Accepted to Generated Ratio* defines the smallest ratio of accepted solutions to total solutions allowed (page C.4). Variation of this parameter did not significantly influence the optimization of Problem One. For problems with little curvature near the optimum, there would be relatively few acceptances compared to generated solutions, and this parameter could have a significant effect.

- *Cost Precision* defines the smallest difference considered significant between cost (weight) values when counting cost repeats of the objective function for *Maximum Cost Repeats* (page C.5). For instance, a cost of 3.01 may be considered the same as 3.00 if *Cost Precision* is set to a value greater than 10^{-2}. In Problem One, a change in cost precision did not make a significant difference in the optimization. This parameter could be significant when very fine precision is required in finding the optimum, and generally this parameter can be set for the precision needed in the solution.

- *Maximum Cost Repeats* stops the optimization when the cost value of the objective function is repeated a given total number of times (page C.6). The change in this value was not significant in this problem. In other problems, extra cost repeats caused the algorithm to "spin its wheels" similar to the effect of extra acceptances. This value can be set at a relatively low value, and increased if necessary for a better solution.

- *Number of Cost Samples* tells the algorithm how many samples to use when initially evaluating the "surface" of the cost function (page C.7). The more samples the algorithm uses, the better the initial survey of the cost function. In Problem One, an increase by a factor of 2.5 in the *Number of Cost Samples* decreased the time required to find the solution by a factor of more than ten. This is a significant improvement.

- *User Initial Parameters* allows the user to give the algorithm an initial feasible solution from which to begin the search (page C.8). If an initial feasible solution is provided, the algorithm centers the search on this value and searches in each direction from this point. If no initial feasible solution is provided, the algorithm searches the neighborhood defined by the upper and lower bounds until a feasible solution is found. In this problem, the algorithm required about 10 percent fewer generated

solutions when it found its own initial feasible solution, and the difference in time was proportionally smaller. Intuitively, the algorithm should find the optimum faster with a good initial feasible solution. However, because of the random nature of the algorithm's search, this is not always the case. In runs where an infeasible solution was given as the starting point, the algorithm generally could not find the feasible region. For these reasons, consideration should be given to setting *User Initial Parameters* to false.

- *Activate Reanneal* allows the algorithm to reach a stopping point, then re-scale the parameter temperatures and begin the annealing again (page C.9). Normally this is done to improve the quality of the solution by reducing round-off error. In Problem One, switching this option to *true* did not change the optimal solution found, but it reduced the number of solutions generated by a factor of 0.5, and reduced the solution time proportionately. This result opens the door to both improved quality of solution and improved solution times.

- The *Upper/Lower Bounds* define the neighborhood initially searched by the algorithm (page C.10). In this case, the optimal solution was at (0.55, 0.10). When the lower and upper bounds were set to zero and ten, respectively, the algorithm found the optimum. When the upper bound was changed to twenty, the number of solutions generated increased by a factor of almost two and the time required increased proportionately. When the upper bound was increased to 100, the algorithm could not find a feasible solution. This is because it was searching such a large space compared to the feasible region. In this case, the probability of finding a feasible solution via random search is very small. This problem is significant whenever the neighborhood defined by the bounds is much larger than the feasible region defined by the constraints. The problem of a large neighborhood size compared to the feasible region became significant for Benchmark Problems Seventeen and Eighteen. This issue is discussed in more detail in section 4.3 (Dimensionality and Constraints).

- Benchmark Problem Four was given negative values as a variation of its lower bounds (pages C.15 and C.16). Normally, structural optimization problems do not allow for negative values of the decision variables. However, to allow for a case where a

comparison (i.e. right wing/left wing) could be expressed as positive/negative values of a variable, the algorithm can find a solution.

- The *Seed Value* for the random number generator influences the Uniform Random Variable value which the algorithm compares with the temperature (page C.10) and so, the number of acceptances. In this case, the algorithm found the solution much more quickly with one of the alternate seed values and much more slowly with the other alternate seed value. In each case, the global optimum was found, but the time varied widely. The effect seems to be unpredictable. This illustrates that the seed value for the random number generator can impact the success of the optimization, and seed values should be tested carefully before use. This is normally done by the originator of a random number generator.

Other variations in the initial conditions/stopping criteria are possible, but weren't considered significant in this application. For instance, the *Include Integer Parameters* option would certainly influence the optimization, but this research is not concerned with integer programming, so no effort was made to evaluate this effect. The complete set of initial conditions/stopping criteria is listed in Appendix B.

4.2 Quality of Solutions Found

For the problems that the Adaptive Simulated Annealing algorithm was able to solve, the quality of solution was excellent compared to the other optimizers. In the worst case, the Adaptive Simulated Annealing algorithm's solution was two percent higher than the best of the other optimizers' solutions. But in all other cases, the algorithm found the best solution or one of the best solutions. Table 4.2 lists the solutions found by the Adaptive Simulated Annealing algorithm and by the other four optimizers. The values given in Table 4.2 for the Adaptive Simulated Annealing solutions are the best found among all the runs for each problem. These solutions were not generally found with the standard conditions. Rather, the initial conditions and stopping criteria were modified as necessary to find the best combination for each problem.

All five optimizers found comparable answers to Problems One through Six. ASA, FUNOPT, ADS, and NLPQL found comparable answers for Problems Seven through Sixteen, although ASA's solution to Problem Twelve is about two percent higher than the best of the other optimizers.

ASA found solutions to the highly constrained problems (Seventeen and Eighteen) only after the bounds on the problem were reduced to a very small part of the original neighborhood, and centered on the optimal solutions found by the other optimizers. This is discussed further in Section 4.3, but analysis of the quality of the solution must consider this fact. Even with the reduced bounds, the solution ASA found for Problem Seventeen has a fifteen percent higher weight than the best solution found by the other optimizers. ASA's solution to problem eighteen (with the modified bounds) compares favorably with the solutions found by the other optimizers.

It should be noted that some of the variation between solutions found by the optimizers can be attributed to the fact that the solution space, near the optimum, may be almost flat. In this case, it is relatively expensive to continue a search near the optimum compared to the potential benefit, so most algorithms stop when "close enough" to the optimum. In this investigation, differences between solutions found by the various optimizers were considered noteworthy if they were larger than one percent.

Alternate Optimal Solutions. Benchmark Problem Seven illustrates the property of alternate optimal solutions, found by varying the initial feasible solution (pages C.20 and C.21). The optimization was started at various initial feasible solutions and in each case a different optimal point was found. The objective function value was the same for all of these points.

Problems Nineteen and Twenty came from a different problem set than the first eighteen, and the other four optimizers were not used to solve them (Floudas and Pardalos, 1990). ASA found the global optimum for Problem Nineteen (under Standard Conditions). It could not find any solution for Problem Twenty, even when the neighborhood was reduced to 10^{-20} of the original neighborhood and centered on the solution provided by Floudas and Pardalos.

Table 4.2 Quality of ASA Solutions Compared to Other Optimizers

Prob	ASA	FUNOPT	ADS	NLPQL	NEWSUMT
1	5.606	5.61	5.61	5.61	5.61
2	1.509	1.514	1.504	1.508	1.519
3	18.474	18.69	18.46	18.47	18.46
4	1.340	1.340	1.340	1.340	1.341
5	1560.129	1560.17	1560.10	1560.06	1560.64
6	2994.31	2997.61	2997.17	2994.89	2990.73
7	*112500	112500	112500	112500	**
8	*10250	10250	10250	10250	**
9	0.200	0.200	0.232	0.200	**
10	0.503	0.500	0.499	0.500	**
11	-1.00	-0.979	-1.00	-1.00	**
12	53.419	52.767	52.055	52.000	**
13	-8.333	-8.332	-8.333	-8.333	**
14	94.164	93.180	93.277	93.254	**
15	3.500	3.500	3.500	3.500	**
16	9.037	9.521	9.036	9.037	**
17	*1042.77	**	912.26	911.88	**
18	*3.954	3.928	3.989	3.951	**
19	-15.0	**	**	**	**
20	No Solution	**	**	**	**

*Modified bounds, **Not solved by this optimizer

4.3 Dimensionality and Constraints

Dimensionality. High dimensionality and high degree of constraint were both significant factors in determining the success of the Adaptive Simulated Annealing algorithm in solving the benchmark problems. "From the complexity point of view global optimization problems belong to the class of NP-hard problems. This means that as the input size of the problem increases the computational time required to solve the problem is expected to grow exponentially" (Floudas and Pardalos, 1990:2). This is because the search must expand in a new dimension representing each variable added, while retaining all of the previous dimensions in the search. For example, in finding length, area, and volume; each added dimension is multiplied by the previous dimensions. As the search proceeds from length, to area, to volume; the number of dimensions increases from one, to two, to three; and the "size" increases from first power to second power, to third power. For higher dimensions the "size" increases to fourth power, and fifth power, and so on. Using a random search, this increase in dimensionality eventually overwhelms the capacity of the algorithm.

Non-linearity of Constraints. The non-linearity and number of constraints in each of the benchmark problems were also significant factors in determining the success of the algorithm in solving the problems. In a problem with many or highly non-linear constraints, the feasible region may be a small part of the neighborhood defined by the upper and lower bounds on each variable. In a highly constrained problem, a random search of the neighborhood is unlikely to find the feasible region. To help identify this effect, the problems which could not be solved with the original bounds were modified. For Problem Seventeen, the neighborhood had to be reduced to approximately 10^{-6} of the original neighborhood and centered near the optimum before the algorithm could find a solution. For Problem Eighteen, the reduced neighborhood was approximately 10^{-8} of the original neighborhood for the algorithm to find a solution. For Problem Twenty, the neighborhood was reduced to approximately 10^{-20} of the original, and the algorithm still could not find a feasible solution. Note that the bounds on these problems could not have been reduced this way unless the optimal solutions were known in advance, so the algorithm wasn't really solving the problems.

4.3.1 Measure of Complexity. Benchmark Problems Seventeen and Eighteen have high levels of constraint and Benchmark Problems Nineteen and Twenty have high dimensionality. It is difficult to isolate the effect of a high level of constraint from the effect of high dimensionality in the results of this investigation because these two effects interact. In an attempt to combine and quantify these effects, a Measure of Complexity (MOC) was calculated for each of the benchmark problems. This is a weighted sum which considers the level of non-linearity of the constraints and objective function, the total number of constraints, and the number of variables in each problem (Section 3.4.4). These factors, as well as the solution time for each benchmark problem are listed in Table 4.3.

Standard Conditions. To validate the solution times reported in Table 4.3, all of the problems were initially optimized under Standard Conditions. Some of the problems could not be solved under the Standard Conditions, and these are highlighted in the table. For these problems, the Standard Conditions were modified as little as possible to allow the algorithm to find a solution. (The Standard Conditions are listed in Appendix B.)

4.3.2 Analysis of the Effects. The results of optimizing each problem are presented in Table 4.3. The Adaptive Simulated Annealing algorithm solved problems with Measure of Complexity (MOC) values below 30 quickly and accurately. All of the problems with Measure of Complexity (MOC) values of 20 and below were solved in less than eleven seconds CPU-time, under the Standard Conditions. All of the problems with MOC values of 30 or less were solved in less than thirty seconds CPU-time, also under Standard Conditions.

Problems having MOC values between 55 and 106 are in a transition zone. The algorithm was able to find solutions, but with some difficulty. Problems Six and Sixteen have MOC values of 55 and 73 respectively. They were solved in less than one minute, but only after modifications to the Standard Conditions. Problem Six required more *Invalid Generated States* and Problem Sixteen required a modified bound. For these problems, knowing the optimal solution in advance was not required to make the modification to the Standard Conditions. For problems Seventeen and Eighteen, with MOC values of 106 and 82 respectively, the bounds had to be modified significantly before a solution could be

Table 4.3 Effect of Dimensionality and Constraints on Solution Time

Prob	Number Vars	Linear Const	Non-Lin Const	Meas of Complex	Solution Time
1	2	2	1	22	0h 0m 38.4s
2	2	0	2	19	0h 0m 1.77s
3	2	0	2	17	0h 0m 1.84s
4	5	0	1	28	0h 0m 22.32s
5	3	0	1	19	0h 0m 3.52s
6	7	0	11	73	0h 0m 58.84s*
7	2	1	2	20	0h 0m 10.15s
8	2	2	0	14	0h 0m 10.43s
9	2	1	0	14	0h 0m 2.56s
10	2	1	0	14	0h 0m 2.1s
11	2	1	0	15	0h 0m 0.66s
12	3	2	0	23	0h 0m 2.92s
13	1**	0**	0**	10	0h 0m 0.61s
14	3	0	2	27	0h 0m 3.14s
15	2	0	2	14	0h 0m 1.76s
16	4	0	3	55	0h 0m 8.76s*
17	7	0	6	106	0h 8m 30.63s*
18	8	0	6	82	0h 24m 29.65s*
19	13	9	0	97	11h 57m 12.7s
20	20	10	0	258	No Solution

*Not solved under Standard Conditions, ** Equality constraint

found. This could have been done only with extensive trial-and-error or by knowing the solution in advance. Problem Nineteen (MOC 97) was solved without modification, under the Standard Conditions, but the solution time was nearly 12 hours of CPU-time.

Problem Twenty has a MOC of 258, and was found to be beyond the capacity of the algorithm. Even with extensive modification to the standard conditions, and with the bounds narrowed to 10^{-20} of the original neighborhood and an initial solution provided near the optimum, the algorithm was unable to find a solution.

4.4 Overall Evaluation

The benchmark problems in this set explored the full range of ability of the Adaptive Simulated Annealing algorithm in both dimensionality and level of constraint. Overall, the Adaptive Simulated Annealing algorithm found excellent solutions to seventeen of the twenty benchmark problems. For the low dimension, lightly constrained problems, the performance of the algorithm is very good. For these problems with MOC values below 50, the solutions found were as good as the solutions found by the other optimizers and the time required was insignificant.

The performance of the algorithm on problems with moderate MOC values was not as good. In some cases, to solve problems with MOC values between 50 and 100, only the initial conditions/stopping criteria had to be modified, and knowledge of the solution was not necessary in advance. Generally, for problems with MOC values between 50 and 100, the optimizer may find a solution with some "tinkering", or may require hours of CPU-time.

In some problems with higher MOC values, the bounds had to be modified significantly and the optimum had to be known before the algorithm could find a solution. This result suggests a possible use for fine tuning a solution in a neighborhood found by some other optimizer. However, the solution found for problem seventeen was not nearly as good as the other optimizers' solutions, suggesting that even this limited application may be undesirable for problems with MOC values above 100. For the problem with a MOC value above 200, the optimizer was unable to find solution.

Analysis of these results was used to draw conclusions about the effectiveness of using Simulated Annealing in structural optimisation problems. Chapter Five summarises the entire research process; addresses the conclusions drawn from the results of the investigation; and makes suggestions for further research relating to Simulated Annealing, non-linear programming, and structural optimization.

V. Summary, Recommendations, and Conclusion

5.1 Summary of the Research

This research was conducted to investigate the advantages and disadvantages of using Simulated Annealing to solve structural optimization problems. Simulated Annealing has been applied to many types of optimization problems, but not to structural optimization until this research was conducted. To determine the applicability of Simulated Annealing to structural optimization problems, the Adaptive Simulated Annealing algorithm was used to optimize twenty benchmark problems. The success or failure of the algorithm in solving these benchmark problems is the basis for evaluation.

5.1.1 Highlights of the Research. Eighteen of the twenty benchmark problems represent "real" structural optimization problems. For research in structural optimization, this is the most valid type of problem, but the largest of these problems only had eight variables. To test the ability of the Simulated Annealing algorithm in higher dimensions, two high-dimension non-linear programs were added to the set of structural optimization problems. These two problems are not "real" structural optimization problems, but they were formulated in a similar manner to the structural optimization problems, and thus allowed the investigation to extend to what may be considered the limits of the algorithm's capability.

Variations in Initial Conditions/Stopping Criteria. Variations in the initial conditions and stopping criteria can greatly impact the success of any optimization. The first benchmark problem was used to test the effects of variations in the initial conditions and stopping criteria. The algorithm found the same solution under each variation, so the effect of these variations was evaluated by comparing the amount of time required to solve the problem under each variation. Based on this evaluation, a set of "Standard Conditions" was established for use with the remaining problems.

Dimensionality and Constraint. To evaluate the effects of dimensionality and constraint, all of the benchmark problems were optimized under the Standard Conditions. Dealing with problems having many variables was expected to be a weakness of this algorithm. However, as the research progressed it became clear that the constraints imposed

on the solution were also significant in determining the success of the algorithm. The constraints may make the feasible region very small compared to the neighborhood searched by the algorithm. The number of variables and the level of non-linearity of the constraints in a problem determine how difficult the problem is to solve using Simulated Annealing. A problem containing many variables, many constraints, and/or highly non-linear constraints is difficult for this algorithm to optimize. It is difficult to separate and assess the effects of dimensionality and constraint because these effects interact.

Measure of Complexity. To help quantify the effects of dimensionality and constraint, a Measure of Complexity (MOC) index was established. This measure is used to assign a relative score to each non-linear program, based on the non-linearity of the objective function and constraints, the number of variables, and the number of constraints. It assigns a single-value score which quantifies the overall topology of the problem, giving a relative indication of how difficult the problem may be to solve.

Quality of Solution. The quality of the solutions found by an algorithm may be the most important criteria for evaluation. In this investigation the algorithm minimized the value of the objective function, and the lowest value which met the constraints was the best solution. The best solutions found by the Adaptive Simulated Annealing algorithm were compared to the best solutions found by four other optimizers currently used in structural optimization. The best solutions found by the Adaptive Simulated Annealing algorithm were found by varying the initial conditions and stopping criteria, as necessary, for each individual problem.

5.1.2 Results. The Adaptive Simulated Annealing algorithm found solutions equivalent to the other optimizers for seventeen of the twenty benchmark problems. The solution time was nearly twelve hours for one of these optimizations, but was insignificant for the other sixteen successful optimizations. The initial conditions were modified as necessary for the algorithm to find the best solutions to these problems, but advance knowledge of the solution was not required to make these modifications.

The algorithm was unable to solve three of the twenty problems. It eventually found solutions to two of these, but only after the bounds on the variables were modified sig-

nificantly and an answer very close to the optimum was provided as an initial feasible solution. Because these modifications could only have been made with advance knowledge of the solution, the algorithm didn't really solve these problems. The algorithm was unable to find a solution to one problem even after modification of the initial conditions, extensive modification of the bounds, and an initial feasible solution very near the optimum.

In these problems, the Measure of Complexity value is a useful tool for predicting the success of the algorithm. MOC values below fifty predict a good solution, very little CPU-time, and few modifications to the initial conditions/stopping criteria. Values between 50 and 150 predict a good solution only with modifications to the initial conditions/stopping criteria, and/or hours CPU-time. MOC values well above 150 predict that the algorithm will not solve the problem.

5.1.2.1 Most Significant Result. Identification of the limitations of the algorithm in solving constrained problems is the most significant result of this research. Previous investigations suggested that high dimensionality might limit the use of the algorithm, but limitations on the level of constraint had not been addressed in the literature.

5.1.3 Evaluation of Applicability. The purpose of this research was to determine if Simulated Annealing is a good tool for structural optimization problems. In light of the results, it appears that Simulated Annealing has only limited applicability in structural optimization. "Real" structural optimization problems may have hundreds, or thousands, of variables and many highly non-linear constraints. Simulated Annealing seems best suited to low dimension, lightly constrained problems, including those with highly non-linear objective functions.

5.2 Recommendations for Further Research

In large structural optimization problems, the number of constraints is normally reduced by an ***active-constraint*** strategy. Also, the number of variables may be reduced by ***linking variables*** together in sets that then work as a single unit (Kamat, 1993:6). Under these circumstances, where the effective MOC value is reduced, Simulated Annealing may

perform as well as, or better than, other optimizers. Further research could establish the conditions where an active-constraint strategy and linking variables would make Simulated Annealing effective in structural optimization.

Measure of Complexity Index. The level of constraint and number of variables are critical to the success of Simulated Annealing, but also are significant for other optimizers. No theoretical basis was established for using the Measure of Complexity in this research; it seems to work, so it was used. It assigns a single quantitative value to the overall topology of a non-linear program. With further research, this measure could provide common ground for comparing the performance of optimizers with significant method-to-method differences. Theoretical proof of the validity of the method would help establish it as a basis for comparison. Consideration should be given for weighting periodic functions, transcendentals, and other non-polynomials.

Parallel Processing. A significant part of the time required to evaluate each new solution proposed in the Simulated Annealing algorithm is used to check feasibility. After a solution is proposed, the variable values are substituted into each constraint. The constraints are evaluated sequentially and, if all of them are met, the objective function is evaluated. After a solution is proposed, a parallel processor could check all of the constraints at once. This would significantly reduce the time required for the calculations. A parallel processor might also be used to choose the variable values for the new proposed solution. In this way, all of the new variables could be selected at once, then the new proposed solution could be evaluated.

Decomposition. A non-linear program with many variables could be partitioned (in a process like Rosen's partitioning algorithm or Benders' decomposition) into multiple sub-problems and a master problem, with each sub-problem having fewer dimensions than the initial problem (Lasdon, 1970:358). The Simulated Annealing process could be used to solve the subproblems, and perhaps the master. The MOC index could be used to guide the partitioning process, breaking the problem up into manageable pieces for SA to solve.

Reannealing. One result of the investigation is that the reannealing process may improve the speed of the Adaptive Simulated Annealing algorithm, along with the expected

result of improving the accuracy of the solution. Analysis of the reason for this behavior, and further exploitation of this advantage would be useful in the development of Adaptive Simulated Annealing in many applications. Dr. Ingber has expressed an interest in supporting further research in this area.

It also seems possible that the algorithm may be able to reduce the actual upper and lower bounds on variables, to reduce the search neighborhood. If this could be done in conjunction with reannealing (or quenching), it may further speed the solution. A related idea would be to partition the solution space, and search in each partition separately. In either case, provisions would have to be made to ensure that the global optimum would be retained within the modified bounds.

Fine-Tuning of Solutions. Further research might identify the circumstances where Simulated Annealing could be used for fine-tuning solutions found by other optimizers.

Random Number Seed. Further research may reveal ways to improve the quality of the solution or the solution time by modifying the random number seed.

Optimizing Initial Conditions. Adaptive Simulated Annealing has an option for self-optimizing the initial conditions of a subset of the non-linear program, then providing these initial conditions for the full optimization. This is a process of annealing the annealing process and is considered to be CPU-intensive. Perhaps a linear model of the effects of variations in the initial conditions or an application of the MOC index could be used to predict the best initial conditions for a given non-linear program. This prediction could be used to provide initial conditions for a full optimization.

5.3 Conclusion

The applicability of Simulated Annealing to structural optimization problems appears to be limited. The effects of high dimensionality and high levels of constraint overwhelm the random search process of this algorithm. While this algorithm is not well suited to this application, it shows promise for further development in many other fields. As computer hardware becomes more capable, the applicability of this algorithm becomes broader and further research is justified.

Appendix A. *Making ASA Run*

The ASA algorithm is set up to make the *user.c* file a template for the user to modify for the individual nonlinear programming (NLP) problem.

a. The user modifies the options as necessary, or allows the algorithm to run the default values.

b. The user defines the number of parameters (dimensions) in the NLP. This allocates storage space for the parameter arrays, and defines D in the *Temperature Ratio Scale* function.

c. The user sets upper and lower bounds for each parameter. This step is required for each parameter, even if the NLP does not define the bounds. ASA uses these values as B and A in the *Parameter Generating PDF*. The values of each parameter are chosen from this range.

d. The user defines constraints, if applicable. These can be built into the algorithm with an IF statement (i.e. if constraint x is met), prior to evaluating the objective function.

e. The user defines the objective function. In *user.c* the value of "summ" is returned to the optimizer, so the objective function is set equal to "summ". The objective function and constraints must be explicitly defined for ASA.

In UNIX, the user can use the "make" command. This command compiles, links, and runs all the necessary files according to the Makefile command (provided with the ASA code). The important files are:

a. *user.c* which the user modifies with options, constraints, and objective function.

b. *user.h* which contains the header files for *user.c*

c. *asa.c* which contains the SA optimizer

d. *asa.h* which contains the header files for *asa.c*

e. *asaopt* (if used) which defines the options chosen

f. *asaout* which is the default output file

Appendix B. *Standard Conditions for ASA*

Standard Adaptive Simulated Annealing initial conditions and stopping criteria for all benchmark problems:

```
OPTIONS_FILE = 0
HAVE_ANSI = 1
IO_PROTOTYPES = 1
TIME_CALC = 1
INT_LONG = 1
INT_ALLOC = 0
SMALL_FLOAT = 1e-18
MIN_DOUBLE = 1e-18
MAX_DOUBLE = 1e+18
EPS_DOUBLE = 1e-18
NO_PARAM_TEMP_TEST = 0
NO_COST_TEMP_TEST = 0
SELF_OPTIMIZE = 0
OPTIONAL_DATA = 0
ASA_PRINT = 1
ASA_OUT = asa_out
USER_ASA_OUT = 0
ASA_PRINT_INTERMED = 1
ASA_PRINT_MORE = 0
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-08
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->TEMPERATURE_RATIO_SCALE = 1e-05
OPTIONS->COST_PARAMETER_SCALE = 1
OPTIONS->TEMPERATURE_ANNEAL_SCALE = 100
```

Standard Bounds: 0 lower, 20 upper

Standard Conditions Continued:

```
OPTIONS->USER_INITIAL_COST_TEMP = 0
OPTIONS->INCLUDE_INTEGER_PARAMETERS = 0
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->INITIAL_PARAMETER_TEMPERATURE = 1
OPTIONS->RATIO_TEMPERATURE_SCALES = 0
OPTIONS->USER_INITIAL_PARAMETERS_TEMPS = 0
OPTIONS->TESTING_FREQUENCY_MODULUS = 100
OPTIONS->ACTIVATE_REANNEAL = 0
OPTIONS->REANNEAL_RESCALE = 10
OPTIONS->MAXIMUM_REANNEAL_INDEX = 50000
OPTIONS->DELTA_X = 0.001
OPTIONS->DELTA_PARAMETERS = 0
OPTIONS->CURVATURE_0 = 0
OPTIONS->QUENCH_PARAMETERS = 0
OPTIONS->QUENCH_COST = 0
```

Appendix C. *Benchmark Problems and ASA Solutions*

```
BENCHMARK PROBLEM #1

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 33453, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 21.53s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED LIMIT ACCEPTANCES

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 10000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 769257, *number_accepted = 1045
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 8m 33.63s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED LIMIT INVALID GEN STATES

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 10000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 33453, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 21.48s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

BENCHMARK PROBLEM #1, MODIFIED ACCEPTED TO GENERATED RATIO

```
Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 1e-08
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 33453, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 21.47s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED COST PRECISION

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 0.0001
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              10              1           -1
 1        0              10              1           -1

Results:
number_generated = 33453, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 21.33s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED MAX COST REPEATS

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 5
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 33453, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 22.31s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED NUMBER COST SAMPLES

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 5
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 3155, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 1.99s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED USER INIT PARAMS (FALSE)

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 0
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 29756, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 20.09s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED ACTIVATE REANNEAL (TRUE)

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 1

index_v param_minimum param_maximum param_value param_type
 0        0             10              1           -1
 1        0             10              1           -1

Results:
number_generated = 14912, *number_accepted = 900
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 9.15s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED SEED VALUES

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              10              0           -1
 1        0              10              0           -1

Results:
number_generated = 3735, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 2.55s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED BOUNDS (0 - 20)

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              20              1           -1
 1        0              20              1           -1

Results:
number_generated = 57787, *number_accepted = 1001
best_generated_state->parameter[0] =    0.5505103
best_generated_state->parameter[1] =    0.1010205

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 5.606123
asa_end:time: 0h 0m 38.4s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM #1,  MODIFIED BOUNDS (0 - 100)

Constraints:
g1 = (-2 * x[0] + x[1] + 1);
g2 = (-x[0] + 2 * x[1] -1);
g3 = (x[0] * x[0] - 2 * x[0] - 2 * x[1] + 1);

Objective Function:
summ = (10 * x[0] + x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              100             1           -1
 1        0              100             1           -1

Results:
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =            0
best_generated_state->parameter[1] =            0

TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 0.23s; incr: 0h 0m 0s

BEST COST WITH MODIFIED PARAMETERS 5.606123
```

```
BENCHMARK PROBLEM  #2, STANDARD CONDITIONS

Constraints:
g1 = (.124 * sqrt(1 + x[1] * x[1]) * (8/x[0] +
1/(x[0] *x[1]) - 1);

g2 = (.124 * sqrt(1 + x[1] * x[1]) * (8/x[0] -
1/(x[0] *x[1]) - 1);

Objective Function:
summ = (x[0] * sqrt(1+ x[1] * x[1])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0.2                4             4             -1
 1        0.1                1.6           1.6           -1

Results:
number_generated = 3482, *number_accepted = 1001
best_generated_state->parameter[0] =      1.419945
best_generated_state->parameter[1] =       0.3610655

 NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 1.509699
asa_end:time: 0h 0m 1.77s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS 1.508653
```

BENCHMARK PROBLEM #3, STANDARD CONDITIONS

```
Constraints:
g1 = -1 + 16/(x[1] + 0.25 * x[0]);
g2 = -1 + sqrt(3)/(3 * x[0]) + 2/(x[1] + 0.25 * x[0]);

Objective Function:
summ = 4.0 * x[0] + x[1]

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0                20               1      -1
 1        0                20               1      -1

Results:
number_generated = 3589, *number_accepted = 1001
best_generated_state->parameter[0] =    0.6598288
best_generated_state->parameter[1] =     15.83505

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 18.47436
asa_end:time: 0h 0m 1.84s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS 18.47436
```

```
BENCHMARK PROBLEM #4, STANDARD CONDITIONS

Constraint:
g1 = 61/ pow(x[0],3) + 37/ pow(x[1],3) + 19/pow(x[2],3) +
 7/pow(x[3],3) + 1/pow(x[4],3) -1;

Objective Function:
summ = 0.0624 * (x[0] + x[1] + x[2] + x[3] + x[4])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL

index_v param_minimum param_maximum param_value param_type
    0             0             20            1          -1
    1             0             20            1          -1
    2             0             20            1          -1
    3             0             20            1          -1
    4             0             20            1          -1

Results:
*number_generated = 17831, *number_accepted = 1001
best_generated_state->parameter[0] =     6.019731
best_generated_state->parameter[1] =     5.236718
best_generated_state->parameter[2] =     4.539208
best_generated_state->parameter[3] =     3.547769
best_generated_state->parameter[4] =     2.134679

 NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 1.340234
asa_end:time: 0h 0m 22.32s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 1.340036
```

```
BENCHMARK PROBLEM #4, NEGATIVE LOWER BOUND

Constraint:
g1 = 61/ pow(x[0],3) + 37/ pow(x[1],3) + 19/pow(x[2],3) +
 7/pow(x[3],3) + 1/pow(x[4],3) -1;

Objective Function:
summ = 0.0624 * (x[0] + x[1] + x[2] + x[3] + x[4])

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL

index_v param_minimum param_maximum param_value param_type
    0           -20           20          1         -1
    1           -20           20          1         -1
    2           -20           20          1         -1
    3           -20           20          1         -1
    4           -20           20          1         -1

Results:
*number_generated = 16548, *number_accepted = 1001
best_generated_state->parameter[0] =     1.286e-14
best_generated_state->parameter[1] =     1.286e-14
best_generated_state->parameter[2] =     1.286e-14
best_generated_state->parameter[3] =     1.286e-14
best_generated_state->parameter[4] =     1.286e-14

 NORMAL_EXIT exit_statu
final_cost = best_gener      state->cost = -6.24
asa_end:time: 0h 0m 15.74s, incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 1.340036
```

```
BENCHMARK PROBLEM # 5, STANDARD CONDITIONS

Constraint:
g1 = -x[0]*x[1]*x[2] + 125;

Objective Function:
summ = 20*x[1]*x[2] + 30*x[0]*x[2] + 15*x[0]*x[1];

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v parameter_minimum parameter_maximum param_value param_type
 0        0                  20                 1           -1
 1        0                  20                 1           -1
 2        0                  20                 1           -1

Results:
number_generated = 6342, *number_accepted = 1001
best_generated_state->parameter[0] =     4.72971
best_generated_state->parameter[1] =     6.53976
best_generated_state->parameter[2] =     4.041359

 COST_REPEATING exit_status = 0
final_cost = best_generated_state->cost = 1565.991
asa_end:time: 0h 0m 3.52s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 1560.129
```

BENCHMARK PROBLEM # 6, MODIFIED INVALID GENERATED STATES

Constraints:

```
g1 = 27.0/(x[0] * pow(x[1],2) * x[2]) -1;
g2 = 397.5/(x[0] * pow(x[1],2) * pow(x[2],2)) -1;
g3 = 1.93 * pow(x[3],3)/(x[1] * x[2] * pow(x[5],4));
g4 = 1.93 * pow(x[4],3)/(x[1] * x[2] * pow(x[6],4));
g5 = sqrt(555025.0 * pow(x[3],2)/(pow(x[1],2) *
 pow(x[2],2) + 16.9E6)/(0.1 * pow(x[5],3)) - 1100.0;
g6 = sqrt(555025.0 * pow(x[4],2)/(pow(x[1],2) *
pow(x[2],2) + 157.5E6)/(0.1 * pow(x[6],3)) -850.0;
g7 = x[1] * x[2] - 40;
g8 = 5 - x[0]/x[1];
g9 = x[0]/x[1]- 12;
g24 = (1.5 * x[5] + 1.9)/x[3] - 1;
g25 = (1.1 * x[6] +1.9)/x[4] - 1;
```

Constraints 10 thru 23 handled by defining bounds (below) on variables

Objective Function:

```
summ = (0.7854 * x[0] * x[1] * x[1]) * (3.3333 *
x[2] * x[2] + 14.9334 * x[2] - 43.0934) - (1.508
* x[0]) * (x[5] * x[5] + x[6] * x[6]) +
7.477 * (x[5] * x[5] * x[5] + x[6] * x[6] * x[6]) +
0.7854 * (x[3] * x[5] * x[5] + x[4] * x[6] * x[6]);
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 10000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	2.6	3.6	3.1	-1
1	0.7	0.8	0.75	-1
2	17	28	22.5	-1
3	7.3	8.3	7.8	-1
4	7.3	8.3	7.8	-1
5	2.9	3.9	3.4	-1
6	5	5.5	5.25	-1

```
Results:
number_generated = 12179, *number_accepted = 1001
best_generated_state->parameter[0] =          3.500005
best_generated_state->parameter[1] =          0.7000001
best_generated_state->parameter[2] =          17
best_generated_state->parameter[3] =          7.300018
best_generated_state->parameter[4] =          7.715332
best_generated_state->parameter[5] =          3.350215
best_generated_state->parameter[6] =          5.286655

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 2994.345
asa_end:time: 0h 0m 58.84s; incr: 0h 0m 0.02s

BEST COST WITH MODIFIED CONDITIONS 2994.31
```

BENCHMARK PROBLEM # 6, STANDARD CONDITIONS

Constraints:

```
g1 = 27.0/(x[0] * pow(x[1],2) * x[2]) -1;
g2 = 397.5/(x[0] * pow(x[1],2) * pow(x[2],2)) -1;
g3 = 1.93 * pow(x[3],3)/(x[1] * x[2] * pow(x[5],4));
g4 = 1.93 * pow(x[4],3)/(x[1] * x[2] * pow(x[6],4));
g5 = sqrt(555025.0 * pow(x[3],2)/(pow(x[1],2) *
 pow(x[2],2) + 16.9E6)/(0.1 * pow(x[5],3)) - 1100.0;
g6 = sqrt(555025.0 * pow(x[4],2)/(pow(x[1],2) *
pow(x[2],2) + 157.5E6)/(0.1 * pow(x[6],3)) -850.0;
g7 = x[1] * x[2] - 40;
g8 = 5 - x[0]/x[1];
g9 = x[0]/x[1]- 12;
g24 = (1.5 * x[5] + 1.9)/x[3] - 1;
g25 = (1.1 * x[6] +1.9)/x[4] - 1;
```

Constraints 10 thru 23 handled by defining bounds (below) on variables

Objective Function:

```
summ = (0.7854 * x[0] * x[1] * x[1]) * (3.3333 *
x[2] * x[2] + 14.9334 * x[2] - 43.0934) - (1.508
* x[0]) * (x[5] * x[5] + x[6] * x[6]) +
7.477 * (x[5] * x[5] * x[5] + x[6] * x[6] * x[6]) +
0.7854 * (x[3] * x[5] * x[5] + x[4] * x[6] * x[6]);
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	2.6	3.6	3.1	-1
1	0.7	0.8	0.75	-1
2	17	28	22.5	-1
3	7.3	8.3	7.8	-1
4	7.3	8.3	7.8	-1
5	2.9	3.9	3.4	-1
6	5	5.5	5.25	-1

```
Results:
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =          0
best_generated_state->parameter[1] =          0
best_generated_state->parameter[2] =          0
best_generated_state->parameter[3] =          0
best_generated_state->parameter[4] =          0
best_generated_state->parameter[5] =          0
best_generated_state->parameter[6] =          0

TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 0.74s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 2994.31
```

```
BENCHMARK PROBLEM # 7, MODIFIED UPPER BOUND (500)

Constraints:
g1 = 2.4E7 - x[0]*x[1]*x[1];
g2 = 1.125E5 - x[0]*x[1];
g3 = x[1] - 2*x[0];

Objective Function:
summ = x[0]*x[1]

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0                500           1            -1
 1        0                500           1            -1

Results:
number_generated = 24267, *number_accepted = 1000
best_generated_state->parameter[0] =     241.9632
best_generated_state->parameter[1] =     464.9468

COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = 112500
asa_end:time: 0h 0m 10.15s; incr: 0h 0m 0s

BEST SOLUTION WITH MODIFIED CONDITIONS 112500
```

```
BENCHMARK PROBLEM # 7, ALTERNATE OPTIMAL SOLUTION

Constraints:
g1 = 2.4E7 - x[0]*x[1]*x[1];
g2 = 1.125E5 - x[0]*x[1];
g3 = x[1] - 2*x[0];

Objective Function:
summ = x[0]*x[1]

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0                500           100           -1
 1        0                500           100           -1

Results:
number_generated = 25161, *number_accepted = 1000
best_generated_state->parameter[0] =     307.3417
best_generated_state->parameter[1] =     366.0421

COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = 112500
asa_end:time: 0h 0m 10.24s; incr: 0h 0m 0.01s

BEST SOLUTION WITH MODIFIED CONDITIONS 112500
```

BENCHMARK PROBLEM # 8, MODIFIED UPPER BOUND

```
Constraints:
g1 = 125.0 - x[0];
g2 = 100.0 - x[1];

Objective Function:
summ = 50 * x[0] + 40 * x[1];

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              500           200           -1
 1        0              500           200           -1

Results:
number_generated = 19248, *number_accepted = 1001
best_generated_state->parameter[0] =          125
best_generated_state->parameter[1] =          100

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 10250
asa_end:time: 0h 0m 10.43s; incr: 0h 0m 0s

BEST SOLUTION WITH MODIFIED CONDITIONS 10250
```

```
BENCHMARK PROBLEM # 9, STANDARD CONDITIONS

Constraint:
g1 = x[0] + 2 * x[1] - 2;

Objective Function:
summ = (x[0] - 1)*(x[0] - 1) + (x[1] - 1)*(x[1] - 1) ;

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             20            1           -1
 1        0             20            1           -1

Results:
number_generated = 5022, *number_accepted = 1001
best_generated_state->parameter[0] =    0.829913
best_generated_state->parameter[1] =    0.5850434

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 0.2011185
asa_end:time: 0h 0m 2.56s; incr: 0h 0m 0.0s

BEST COST WITH MODIFIED CONDITIONS  0.2004642
```

BENCHMARK PROBLEM #10, STANDARD CONDITIONS

Constraints:
g1 = x[0] + x[1] - 5;

Objective Function:
summ = (x[0] - 3) * (x[0] - 3) + (x[1] - 3) * (x[1] - 3) ;

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v	param_minimum	param_maximum	param_value	param_type
0	0	20	1	-1
1	0	20	1	-1

Results:
***number_generated = 4152, *number_accepted = 1001**
best_generated_state->parameter[0] = 2.642822
best_generated_state->parameter[1] = 2.357178

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 0.5407964
asa_end:time: 0h 0m 2.1s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS 0.5026903

```
BENCHMARK PROBLEM # 11, STANDARD CONDITIONS

Constraint:
g1 = x[0] + x[1] - 4;

Objective Function:
summ = (x[0] - 1) * (x[0] - 1) +
(x[1] - 1) * (x[1] - 1) - 2 * x[1] + 2.0 ;

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             20            1           -1
 1        0             20            1           -1

Results:
number_generated = 1711, *number_accepted = 700
best_generated_state->parameter[0] =     1.000001
best_generated_state->parameter[1] =     2


 COST_REPEATING exit_status = 3
final_cost = best_generated_state->cost = -1
asa_end:time: 0h 0m 0.66s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS -1.0
```

BENCHMARK PROBLEM # 12, STANDARD CONDITIONS

```
Constraints:
g1 = 10 - (x[0] + x[1]);
g2 = 8 - (x[1] + 2 * x[2]);

Objective Function:
summ = (x[0] * x[0]) + (x[1]  * x[1]) + (x[2] * x[2]);

Initial Conditons:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0             20              1        -1
 1        0             20              1        -1
 2        0             20              1        -1

Results:
number_generated = 4524, *number_accepted = 1001
best_generated_state->parameter[0] =     5.74947
best_generated_state->parameter[1] =     4.25053
best_generated_state->parameter[2] =       1.874735

 NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 54.63804
asa_end:time: 0h 0m 2.92s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS  53.41885
```

```
BENCHMARK PROBLEM # 13, STANDARD CONDITIONS

Constraint:
x[1] = 4 - x[0];

Objective Function in terms of x[0] only:
summ = 4 * x[0] * x[0] + 3 * (4 - x[0]) * (4 - x[0]) -
5 * x[0] * (4 - x[0]) - 8 * x[0] ;

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0              20              1           -1


Results:
number_generated = 1715, *number_accepted = 1001
best_generated_state->parameter[0] =     2.166665
best_generated_state->parameter[1] =     1.833335

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = -8.333333
asa_end:time: 0h 0m 0.61s; incr: 0h 0m 0.01s

BEST GENERATED COST WITH MODIFIED CONDITIONS -8.333333
```

```
BENCHMARK PROBLEM # 14, STANDARD CONDITIONS

Constraints:
g1 = 16/(x[0] * x[0]) + 16/(x[1] * x[1]) - 1;
g2 = 16/(x[1] * x[1]) + 16/(x[2] * x[2]) - 1;

Objective Function:
summ = (x[0] * x[0]) + (x[1]  * x[1]) + (x[2] * x[2]);

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0


index_v param_minimum param_maximum param_value param_type
 0        0             20              1           -1
 1        0             20              1           -1
 2        0             20              1           -1

Results:
number_generated = 4152, *number_accepted = 1001
best_generated_state->parameter[0] =     5.777124
best_generated_state->parameter[1] =     5.543801
best_generated_state->parameter[2] =     5.77119

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 97.48401
asa_end:time: 0h 0m 3.14s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS 94.16438
```

```
BENCHMARK PROBLEM # 15, STANDARD CONDITIONS

Constraints:
g1 = 1.0/x[0] - 1.0;
g2 = 1.5/(x[1] + x[0]/4) - 1.0;

Objective Function:
summ = x[0] + 2 * x[1] ;

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0

index_v param_minimum param_maximum param_value param_type
 0        0                20           1           -1
 1        0                20           1           -1

Results:
number_generated = 3570, *number_accepted = 1001
best_generated_state->parameter[0] =     2.402833
best_generated_state->parameter[1] =     0.8992917

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 4.201417
asa_end:time: 0h 0m 1.76s; incr: 0h 0m 0s

BEST COST WITH MODIFIED CONDITIONS 3.50
```

BENCHMARK PROBLEM # 16, STANDARD CONDITIONS

Constraints:

```
    g1 = (x[0]*x[0] + x[0]) + (x[1]*x[1] - x[1]) +
(x[2]*x[2] + x[2]) + (x[3]*x[3] - x[3]) - 8.0 ;
    g2 = (x[0]*x[0] - x[0]) + 2*x[1]*x[1] + x[2]*x[2] +
(2*x[3]*x[3] - x[3]) - 10.0 ;
    g3 = (2*x[0]*x[0] + 2*x[0]) + (x[1]*x[1] - x[1]) +
x[2]*x[2] - x[3] - 5.0 ;
```

Objective Function:

```
summ = (x[0]*x[0] - 5*x[0]) + (x[1]*x[1] - 5*x[1]) +
(2*x[2]*x[2] - 21*x[2]) + (x[3]*x[3] + 7*x[3]) + 50.0 ;
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	0	20	1	-1
1	0	20	1	-1
2	0	20	1	-1
3	0	20	1	-1

Results:

```
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =     0
best_generated_state->parameter[1] =     0
best_generated_state->parameter[2] =     0
best_generated_state->parameter[3] =     0

TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 0.43s; incr: 0h 0m 0.0s

BEST COST FOUND WITH MODIFIED CONDITIONS 9.036721
```

BENCHMARK PROBLEM # 16, MODIFIED BOUNDS (0,10)

Constraints:

```
    g1 = (x[0]*x[0] + x[0]) + (x[1]*x[1] - x[1]) +
(x[2]*x[2] + x[2]) + (x[3]*x[3] - x[3]) - 8.0 ;
    g2 = (x[0]*x[0] - x[0]) + 2*x[1]*x[1] + x[2]*x[2] +
(2*x[3]*x[3] - x[3]) - 10.0 ;
    g3 = (2*x[0]*x[0] + 2*x[0]) + (x[1]*x[1] - x[1]) +
x[2]*x[2] - x[3] - 5.0 ;
```

Objective Function:

```
summ = (x[0]*x[0] - 5*x[0]) + (x[1]*x[1] - 5*x[1]) +
(2*x[2]*x[2] - 21*x[2]) + (x[3]*x[3] + 7*x[3]) + 50.0 ;
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	0	10	1	-1
1	0	10	1	-1
2	0	10	1	-1
3	0	10	1	-1

Results:

```
number_generated = 10102, *number_accepted = 1001
best_generated_state->parameter[0] =     0.001839589
best_generated_state->parameter[1] =     1.037515
best_generated_state->parameter[2] =     2.22652
best_generated_state->parameter[3] =     1.068709e-08

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 9.037529
asa_end:time: 0h 0m 8.76s; incr: 0h 0m 0.01s

BEST COST FOUND WITH MODIFIED CONDITIONS 9.036721
```

BENCHMARK PROBLEM #17, MODIFIED BOUNDS (+,- 0.8 UNITS)

Constraints:

```
    g1 = 0.5*sqrt(x[0])*x[6]/(x[2]*x[5]*x[5]) +
0.7*pow(x[0],3.0)*x[1]*x[5]*sqrt(x[6])/(x[2]*x[2]) +
0.2*x[2]*pow(x[5],0.667)*pow(x[6],0.25)/(x[1]*sqrt(x[3])) - 1.0 ;

    g2 = 1.3*x[1]*x[5]/(sqrt(x[0])*x[2]*x[4]) +
0.8*x[2]*x[5]*x[5]/(x[3]*x[4]) +
3.1*sqrt(x[1])*pow(x[5],0.333)/(x[0]*x[3]*x[3]*x[4]) - 1.0;

    g3 = 2.0*x[0]*x[4]*pow(x[6],0.333)/(pow(x[2],1.5)*x[5]) +
0.1*x[1]*x[4]/(sqrt(x[2])*x[5]*sqrt(x[6])) +
x[1]*sqrt(x[2])*x[4]/x[0] +
 0.65*x[2]*x[4]*x[6]/(x[1]*x[1]*x[5]) - 1.0;

    g4 = 0.2*x[1]*sqrt(x[4])*pow(x[6],0.333)/(x[0]*x[0]*x[3]) +
0.3*sqrt(x[0])*x[1]*x[1]*x[2]*pow(x[3],1/3)*pow(x[6],1/4)/pow(x[4],2/3) +
0.4*x[2]*x[4]*pow(x[6],0.75)/(pow(x[0],3)*x[1]*x[1]) +
0.5*x[3]*sqrt(x[6])/(x[2]*x[2]) - 1.0;

    g5 = 10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]) - 3000;

g6 = 100.0 -
(10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]));
```

Objective Function:

```
summ = 10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]);
```

```
Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-08
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	3	4.6	3.873	-1
1	0	1.6	0.801	-1
2	1.8	3.4	2.616	-1
3	3.4	5	4.266	-1
4	0	1.6	0.85	-1
5	0.2	1.8	1.095	-1
6	0	0.8	0.027	-1

```
Results:
number_generated = 56185, *number_accepted = 1001
best_generated_state->parameter[0] =      4.00166
best_generated_state->parameter[1] =      0.5775299
best_generated_state->parameter[2] =      2.380327
best_generated_state->parameter[3] =      3.439459
best_generated_state->parameter[4] =      0.9862509
best_generated_state->parameter[5] =      1.164156
best_generated_state->parameter[6] =      0.001627114

NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = 1042.766
asa_end:time: 0h 8m 30.63s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 936.8749
```

BENCHMARK PROBLEM #17, MODIFIED BOUNDS (+,- 0.9 UNITS)

Constraints:

```
    g1 = 0.5*sqrt(x[0])*x[6]/(x[2]*x[5]*x[5]) +
0.7*pow(x[0],3.0)*x[1]*x[5]*sqrt(x[6])/(x[2]*x[2]) +
0.2*x[2]*pow(x[5],0.667)*pow(x[6],0.25)/(x[1]*sqrt(x[3])) - 1.0 ;

    g2 = 1.3*x[1]*x[5]/(sqrt(x[0])*x[2]*x[4]) +
0.8*x[2]*x[5]*x[5]/(x[3]*x[4]) +
3.1*sqrt(x[1])*pow(x[5],0.333)/(x[0]*x[3]*x[3]*x[4]) - 1.0;

    g3 = 2.0*x[0]*x[4]*pow(x[6],0.333)/(pow(x[2],1.5)*x[5]) +
0.1*x[1]*x[4]/(sqrt(x[2])*x[5]*sqrt(x[6])) +
x[1]*sqrt(x[2])*x[4]/x[0] +
 0.65*x[2]*x[4]*x[6]/(x[1]*x[1]*x[5]) - 1.0;

    g4 = 0.2*x[1]*sqrt(x[4])*pow(x[6],0.333)/(x[0]*x[0]*x[3]) +
0.3*sqrt(x[0])*x[1]*x[1]*x[2]*pow(x[3],1/3)*pow(x[6],1/4)/pow(x[4],2/3) +
0.4*x[2]*x[4]*pow(x[6],0.75)/(pow(x[0],3)*x[1]*x[1]) +
0.5*x[3]*sqrt(x[6])/(x[2]*x[2]) - 1.0;

    g5 = 10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]) - 3000;

g6 = 100.0 -
(10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]));
```

Objective Function:

```
summ = 10.0*x[0]*x[3]*x[3]*pow(x[6],0.125)/(x[1]*pow(x[5],3.0)) +
15.0*x[2]*x[3]/(x[0]*x[1]*x[1]*x[4]*sqrt(x[6])) +
20.0*x[1]*x[5]/(x[0]*x[0]*x[3]*x[4]*x[4]) +
25.0*x[0]*x[0]*x[1]*x[1]*sqrt(x[4])*x[6]/(x[2]*x[5]*x[5]);
```

```
Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-08
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0



index_v param_minimum param_maximum param_value param_type
 0        2.9           4.7           3.873         -1
 1        0             1.7           0.801         -1
 2        1.9           3.5           2.616         -1
 3        3.3           5.1           4.266         -1
 4        0             1.7           0.85          -1
 5        0.1           1.9           1.095         -1
 6        0             0.9           0.027         -1

Results:
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =      0
best_generated_state->parameter[1] =      0
best_generated_state->parameter[2] =      0
best_generated_state->parameter[3] =      0
best_generated_state->parameter[4] =      0
best_generated_state->parameter[5] =      0
best_generated_state->parameter[6] =      0

TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 1.27s; incr: 0h 0m 0.01s

BEST COST WITH MODIFIED CONDITIONS 936.8749
```

BENCHMARK PROBLEM #18, MODIFIED BOUNDS (+,- 0.5 UNITS)

Constraints:

```
g1 = 0.0588*x[4]*x[6] + 0.1*x[0] - 1.0 ;

g2 = 0.0588*x[5]*x[7] + 0.1*x[0] + 0.1*x[1] - 1.0;

g3 = 4.0*x[2]*pow(x[4],-1.0) +
2.0*pow(x[2],-0.71)*pow(x[4],-1.0) +
0.0588*x[6]*pow(x[2],-1.30) - 1.0;

g4 = 4.0*x[3]*pow(x[5],-1.0) +
2.0*pow(x[3],-0.71)*pow(x[5], -1.0) +
0.0588*x[7]*pow(x[3],-1.3) - 1.0;

g5 = 0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) +
10.0 - x[0] - x[1] - 4.2;

g6 = 0.1 - (0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) +
10.0 - x[0] - x[1]);
```

Objective Function:

```
summ = 0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) + 10.0 - x[0] - x[1];
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 0.0001
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	parameter_minimum	parameter_maximum	param_value	param_type
0	5.7	6.7	6.226	-1
1	2	3	2.582	-1
2	0.1	1.1	0.629	-1
3	0.1	1.1	0.663	-1
4	5.4	6.4	5.994	-1
5	5	6	5.541	-1
6	0.5	1.5	1.081	-1
7	0	0.8	0.381	-1

```
Results:
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =      0
best_generated_state->parameter[1] =      0
best_generated_state->parameter[2] =      0
best_generated_state->parameter[3] =      0
best_generated_state->parameter[4] =      0
best_generated_state->parameter[5] =      0
best_generated_state->parameter[6] =      0
best_generated_state->parameter[7] =      0

TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 1.16s; incr: 0h 0m 0.02s

BEST COST WITH MODIFIED PARAMETERS 3.954401
```

BENCHMARK PROBLEM #18, MODIFIED BOUNDS (+,- 0.5 UNITS), AND OTHERS

Constraints:

```
g1 = 0.0588*x[4]*x[6] + 0.1*x[0] - 1.0 ;

g2 = 0.0588*x[5]*x[7] + 0.1*x[0] + 0.1*x[1] - 1.0;

g3 = 4.0*x[2]*pow(x[4],-1.0) +
2.0*pow(x[2],-0.71)*pow(x[4],-1.0) +
0.0588*x[6]*pow(x[2],-1.30) - 1.0;

g4 = 4.0*x[3]*pow(x[5],-1.0) +
2.0*pow(x[3],-0.71)*pow(x[5], -1.0) +
0.0588*x[7]*pow(x[3],-1.3) - 1.0;

g5 = 0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) +
10.0 - x[0] - x[1] - 4.2;

g6 = 0.1 - (0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) +
10.0 - x[0] - x[1]);
```

Objective Function:

```
summ = 0.4*pow(x[0],0.67)*pow(x[6],-0.67) +
0.4*pow(x[1],0.67)*pow(x[7],-0.67) + 10.0 - x[0] - x[1];
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 10000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 10000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 1e-06
OPTIONS->COST_PRECISION = 1e-18
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 5
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	parameter_minimum	parameter_maximum	param_value	param_type
0	5.7	6.7	6.226	-1
1	2	3	2.582	-1
2	0.1	1.1	0.629	-1
3	0.1	1.1	0.663	-1
4	5.4	6.4	5.994	-1
5	5	6	5.541	-1
6	0.5	1.5	1.081	-1
7	0	0.8	0.381	-1

```
Results:
number_generated = 170636, *number_accepted = 2236
best_generated_state->parameter[0] =      6.27728
best_generated_state->parameter[1] =      2.455755
best_generated_state->parameter[2] =      0.6727257
best_generated_state->parameter[3] =      0.5941369
best_generated_state->parameter[4] =      5.96429
best_generated_state->parameter[5] =      5.520324
best_generated_state->parameter[6] =      1.061509
best_generated_state->parameter[7] =      0.3903215

P_TEMP_TOO_SMALL exit_status = 1
final_cost = best_generated_state->cost = 3.954401
asa_end:time: 0h 24m 29.65s; incr: 0h 0m 0.02s

BEST COST WITH MODIFIED PARAMETERS 3.954401
```

BENCHMARK PROBLEM #19, STANDARD CONDITIONS

Constraints:

```
g1 = 2*x[0] +2*x[1] +x[9] +x[10] -10;

g2 = 2*x[0] +2*x[2] +x[9] +x[11] -10;

g3 = 2*x[1] +2*x[2] +x[10] +x[11] -10;

g4 = -8*x[0] +x[9];

g5 = -8*x[1] +x[10];

g6 = -8*x[2] +x[11];

g7 = -2*x[3] -x[4] -x[9];

g8 = -2*x[5] -x[6] -x[10];

g9 = -2*x[7] -x[8] +x[11];
```

Objective Function:

```
summ=5*x[0]+5*x[1]+5*x[2] +
5*x[3]-5.0*(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]+x[3]*x[3])
-x[4]-x[5]-x[6]-x[7]-x[8]-x[9]-x[10]-x[11]-x[12];
```

Initial Conditions:

```
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 1e-04
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

```
index_v param_minimum param_maximum param_value param_type
   0         0              1             1           -1
   1         0              1             1           -1
   2         0              1             1           -1
   3         0              1             1           -1
   4         0              1             1           -1
   5         0              1             1           -1
   6         0              1             1           -1
   7         0              1             1           -1
   8         0              1             1           -1
   9         0             10             3           -1
  10         0             10             3           -1
  11         0             10             3           -1
  12         0              1             1           -1

Results:
number_generated = 13232607, *number_accepted = 1001
best_generated_state->parameter[0] =            1
best_generated_state->parameter[1] =            1
best_generated_state->parameter[2] =            1
best_generated_state->parameter[3] =            1
best_generated_state->parameter[4] =            1
best_generated_state->parameter[5] =            1
best_generated_state->parameter[6] =            1
best_generated_state->parameter[7] =            1
best_generated_state->parameter[8] =            1
best_generated_state->parameter[9] =            3
best_generated_state->parameter[10] =           3
best_generated_state->parameter[11] =           3
best_generated_state->parameter[12] =           1

 NORMAL_EXIT exit_status = 0
final_cost = best_generated_state->cost = -15
asa_end:time: 11h 57m 12.7s; incr: 0h 0m 0.03s

BEST COST WITH MODIFIED CONDITIONS -15
```

BENCHMARK PROBLEM #20, MODIFIED BOUNDS (+/- 0.1 UNITS)

Constraints:

```
    g1 = x[0] +x[1] +x[2] +x[3] +x[4] +x[5] +x[6] +x[7]+
x[8] +x[9] +x[10] +x[11] +x[12] +x[13] +x[14] +x[15] +
x[16] +x[17] +x[18] +x[19] +5;

g2 = -x[0] -x[1] -9*x[2] +3*x[3] +5*x[4] +x[7] +7*x[8] -
-7*x[9] -4*x[10] -6*x[11] -3*x[12] +7*x[13] -5*x[15] +
x[16] +x[17] +2*x[19] -2;

g3 = 2*x[0] -x[1] -x[2] -9*x[3] +3*x[4] +5*x[5] +x[8] +
7*x[9] -7*x[10] -4*x[11] -6*x[12] -3*x[13] +7*x[14] -
5*x[16] +x[17] +x[18] +1;

g4 = 2*x[1] -x[2] -x[3] -9*x[4] +3*x[5] +5*x[6] +x[9] +
7*x[10] -7*x[11] -4*x[12] -6*x[13] -3*x[14] +7*x[15] -
5*x[17] +x[18] +x[19] +3;

g5 = x[0] +2*x[2] -x[3] -x[4] -9*x[5] +3*x[6] +5*x[7] +
x[10] +7*x[11] -7*x[12] -4*x[13] -6*x[14] -3*x[15] +
7*x[16] -5*x[18] +x[19] -5;

g6 = x[0] +x[1] +2*x[3] -x[4] -x[5] -9*x[6] +3*x[7] +
5*x[8] +x[11] +7*x[12] -7*x[13] -4*x[14] -6*x[15] -
3*x[16] +7*x[17] -5*x[19] -4;

g7 =-5*x[0] +x[1] +x[2] +2*x[4] -x[5] -x[6] -9*x[7] +
3*x[8] +5*x[9] +x[12] +7*x[13] -7*x[14] -4*x[15] -
6*x[16] -3*x[17] +7*x[18] +1;

g8 =-5*x[1] +x[2] +x[3] +2*x[5] -x[6] -x[7] -9*x[8] +
3*x[9] +5*x[10] +x[13] +7*x[14] -7*x[15] -4*x[16] -
6*x[17] -3*x[18] +7*x[19];

g9 = 7*x[0] -5*x[2] +x[3] +x[4] +2*x[6] -x[7] -x[8] -
9*x[9] +3*x[10] +5*x[11] +x[14] +7*x[15] -7*x[16] -
4*x[17] -6*x[18] -3*x[19] -9;

g10=-3*x[0] +7*x[1] -5*x[3] +x[4] +x[5] +2*x[7] -x[8] -
x[9] -9*x[10] +3*x[11] +5*x[12] +x[15] +7*x[16] -7*x[17] -
4*x[18] -6*x[19]-40;
```

```
Objective Function:
summ=-0.5*((x[0]-2)*(x[0]-2)+(x[1]-2)*(x[1]-2)+
(x[2]-2)*(x[2]-2)+(x[3]-2)*(x[3]-2)+
(x[4]-2)*(x[4]-2)+(x[5]-2)*(x[5]-2)+(x[6]-2)*(x[6]-2)+
(x[7]-2)*(x[7]-2)+(x[8]-2)*(x[8]-2)+(x[9]-2)*(x[9]-2)+
(x[10]-2)*(x[10]-2)+(x[11]-2)*(x[11]-2)+(x[12]-2)*(x[12]-2)+
(x[13]-2)*(x[13]-2)+(x[14]-2)*(x[14]-2)+(x[15]-2)*(x[15]-2)+
(x[16]-2)*(x[16]-2)+(x[17]-2)*(x[17]-2)+(x[18]-2)*(x[18]-2)+
(x[19]-2)*(x[19]-2));

Initial Conditions:
OPTIONS->LIMIT_ACCEPTANCES = 1000
OPTIONS->LIMIT_INVALID_GENERATED_STATES = 1000
OPTIONS->ACCEPTED_TO_GENERATED_RATIO = 1e-04
OPTIONS->COST_PRECISION = 1e-8
OPTIONS->MAXIMUM_COST_REPEAT = 2
OPTIONS->NUMBER_COST_SAMPLES = 2
OPTIONS->USER_INITIAL_PARAMETERS = 1
OPTIONS->ACTIVATE_REANNEAL = 0
```

index_v	param_minimum	param_maximum	param_value	param_type
0	0	0.1	0	-1
1	0	0.1	0	-1
2	28.7	28.9	28.8	-1
3	0	0.1	0	-1
4	0	0.1	0	-1
5	4	4.2	4.1	-1
6	0	0.1	0	-1
7	0	0.1	0	-1
8	0	0.1	0	-1
9	0	0.1	0	-1
10	0	0.1	0	-1
11	0	0.1	0	-1
12	0	0.1	0	-1
13	0	0.1	0	-1
14	0.5	0.7	0.6	-1
15	3.9	4.1	4	-1
16	0	0.1	0	-1
17	2.2	2.4	2.3	-1
18	0	0.1	0	-1
19	0	0.1	0	-1

```
Results:
number_generated = 0, *number_accepted = 0
best_generated_state->parameter[0] =            0
best_generated_state->parameter[1] =            0
best_generated_state->parameter[2] =            0
best_generated_state->parameter[3] =            0
best_generated_state->parameter[4] =            0
best_generated_state->parameter[5] =            0
best_generated_state->parameter[6] =            0
best_generated_state->parameter[7] =            0
best_generated_state->parameter[8] =            0
best_generated_state->parameter[9] =            0
best_generated_state->parameter[10] =           0
best_generated_state->parameter[11] =           0
best_generated_state->parameter[12] =           0
best_generated_state->parameter[13] =           0
best_generated_state->parameter[14] =           0
best_generated_state->parameter[15] =           0
best_generated_state->parameter[16] =           0
best_generated_state->parameter[17] =           0
best_generated_state->parameter[18] =           0
best_generated_state->parameter[19] =           0


TOO_MANY_INVALID_STATES exit_status = 4
final_cost = best_generated_state->cost = 0
asa_end:time: 0h 0m 2.03s; incr: 0h 0m 0.01s



BEST COST WITH MODIFIED CONDITIONS  none
```

Bibliography

Aarts, E. and J. Korst. ***Simulated Annealing and Boltzmann Machines*** New York: John Wiley and Sons 1990.

Basu, Atanu and L. Neil Fraser. "Rapid Determination of the Critical Temperature in Simulated Annealing Inversion," *Science*, 1409 - 1412 (21 September 1990).

Baker, W. M. Professor of Mathematics, Air Force Institute of Technology, Wright Patterson Air Force Base OH. Personal Interview. 20 January 1994.

Brooks, Daniel G. and William A. Verdini. "Computational Experience with Simulated Annealing Over Continuous Variables," *American Journal of Mathematical and Management Sciences*, 8:425 - 449 (1988).

Canfield Robert A. Associate Professor of Aeronautical Engineering, Air Force Institute of Technology, Wright Patterson Air Force Base OH. Personal Interview. 6 August 1993.

Canfield, Robert A. and V. B. Venkayya. "Implementation of Generalized Optimality Criteria in a Multidisciplinary Environment," *Journal of Aircraft*, 27:1037-1042 (December 1990).

Cerny, V. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *Journal of Optimization Theory and Applications*, 45:41-51 (January 1985).

Collins, Eglese, and Golden. "Simulated Annealing Bibliography," *American Journal of Mathematics and Management Science*, 8:212-233 (1988).

Coulliette, D. Associate Professor of Mathematics, Air Force Institute of Technology, Wright Patterson Air Force Base OH. Personal Interview. 20 January 1994.

Floudas, C.A., and P.M. Pardalos. *Lecture Notes in Computer Science, Number 455, A Collection of Test Problems for Constrained Global Optimization Algorithms* New York:Springer Verlag 1990.

Goldstein, Larry and Michael Waterman. "Neighborhood Size in the Simulated Annealing Algorithm," *American Journal of Mathematical and Management Sciences*, 8:409-423 (1988).

Ingber, Lester. "Simulated Annealing: Practice Versus Theory," *Mathematical Computer Modeling*, 18:29-58 (1993).

Ingber, Lester and Bruce Rosen. "Genetic Algorithms and Very Fast Simulated Annealing," *Mathematical Computer Modeling*, 16:87-100 (1992).

Ingber, Lester. *Adaptive Simulated Annealing (ASA) v1.44, README.ps.* Computer Software. GNU General Public License (GPL), ftp.caltech.edu pub/ingber directory, (24 September 1993).

Kamat M.P. *Structural Optimization Status and Promise.* New York:AIAA Press, 1993.

Kirkpatrick, S., C.D. Gelatt and M.P. Vecchi. "Optimization by Simulated Annealing," *Science*, 220:671-680 (13 May 1983).

Lasdon, Leon S. *Optimization Theory for Large Systems.* New York:McMillan Publishing Co. 1970.

McLaughlin, Michael P. "Simulated Annealing," *Dr. Dobbs Journal*, 26-36 (September 1989).

Tovey. Craig A. "Simulated Simulated Annealing," *Am. Journal of Mathematical and Management Sciences*, 8:389-407 (1988).

Vanderplaats, G. *Numerical Optimization Techniques for Engineering Design.* New York:McGraw Hill, 1984.

Venkayya, V. B. "Bench-Mark Studies Using Various Optimization Algorithms," Conference Report. NATO/DFG Advanced Study Institute. University of Essen, Berchtesgaden, Germany September 23 - October 4 1991.

Venkayya, V.B. and E. Johnson. *Automated Structural Optimization (ASTROS) Theoretical Manual.* Wright Laboratories, Flight Dynamics Directorate, Wright Patterson AFB OH, December 1988.

Venkayya, V.B., V. Tischler, and S. Pitrof. "Benchmarking in Structural Optimization," *AIAA Conference Papers*, 92-4784, September 1992.

Vita

Captain McEachin grew up in a large family in Kalamazoo Michigan. He joined the Air Force and trained as an aircraft weapons mechanic and then an Explosive Ordnance Disposal (EOD) specialist. He was assigned in EOD at Barksdale AFB, Louisiana and Spangdahlem AB, Germany. While assigned to Barksdale AFB, he participated in the Render Safe Procedure and cleanup of the 1980 Titan missile explosion at Damascus Arkansas. He completed his Bachelor's degree at Southern Illinois University while on active duty, and was selected for Officer Training School. After commissioning, he trained as a navigator and Electronic Warfare (EW) Officer and was assigned to the B-52 squadron at Blytheville (renamed Eaker) AFB, Arkansas. While in Arkansas, Captain McEachin worked as an Electronic Warfare Officer, Staff Instructor and Evaluator, and was certified for SIOP alert for over eight years. Captain McEachin participated in Operation Desert Storm on the planning staff of the 806th Bombardment Wing (Provisional) and flew two combat sorties, logging over 26 hours of combat time with the wing. Captain McEachin was selected for an AFIT education in Operations Research, Strategic and Tactical Sciences, and is expecting to be assigned to Headquarters US Strategic Command as a strategic analyst.

Permanent address: 1072 Bayfield Dr.
Beavercreek, Ohio 45430

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Master's Thesis

4. TITLE AND SUBTITLE
AN INVESTIGATION OF SIMULATED ANNEALING APPLIED TO STRUCTURAL OPTIMIZATION PROBLEMS

5. FUNDING NUMBERS

6. AUTHOR(S)
Richard C. McEachin, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GST/ENS/94M-08

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. V. B. Venkayya
Wright Laboratories, Flight Dynamics Directorate
Optimisation Branch, WL/FIBRA
Wright-Patterson AFB, OH 45433-6553

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

This thesis investigates the feasibility of using Simulated Annealing (SA) in structural optimisation problems. The investigation involves solving benchmark structural optimisation problems with an SA algorithm, and comparing its solutions to those found by four other optimisers. Overall, the analysis shows that SA has limited applicability in structural optimisation. Two primary factors were found to adversely impact the performance of the SA algorithm in these problems. These factors are high dimensionality, and high levels of constraint. The difficulty involved in solving these problems with a random search increases exponentially with the number of dimensions. The number, and non-linearity, of the constraints also have an appreciable effect on the success of the algorithm. A Measure of Complexity was created to quantify the combined effect of dimensionality and level of constraint. This measure can be used to predict the applicability of the SA algorithm in optimising a given system of non-linear equations.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Stochastic Processes, Non-Linear Programming, Mathematical Programming, Simulated Annealing, Non-Convex Programming	108
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL